Un framework per l'analisi statica di Deadlock in Java

Vincenzo Lomonaco, Marco Tiseo, Andrea Benetti, Alfred Dhoga

Dipartimento di Informatica, Università di Bologna, Italia.

Abstract. In questo report, riassumiamo brevemente l'attività svolta durante il corso di *Analisi Statica di Programmi* circa la definizione ed implementazione di un framework per l'analisi statica di Deadlock in Java. Dapprima ci si è concentrati sulla definizione di una sintassi ed una grammatica in *ANTLR4* per il bytecode Java e successivamente alla stesura della semantica operazionale del linguaggio. Infine, alla creazione di un sistema di tipi comportamentali per l' estrazione delle dipendenze tra entità utili alla rilevazione di deadlock per il back-end già studiato ed implementato in [2].

1 Introduzione

Nell'ambito del corso di *Analisi Statica di Programmi* abbiamo avuto modo di studiare ed approfondire un framework per l'analisi statica di deadlock in *coreABS*. In [2] viene descritto dettagliatamente l'approccio teorico e metodologico per la rilevazione di deadlock, partendo dalla sintassi specifica di coreABS e giungendo ad una rappresentazione astratta circa le dipendenze tra le entità di un programma. Mediante un sofisticato algoritmo, quindi, è possibile individuare le circolarità rappresentanti la possibilità che si verifichi una situazione di deadlock a runtime. Sulla scia di questo importante lavoro, si è voluto tentare un analogo approccio lavorando direttamente sul bytecode Java e quindi bypassando l'eterogeneità sintattica delle possibili librerie Java dedite al parallelismo. L'obiettivo finale è giungere ad una rappresentazione astratta delle dipendenze identica a quella riportata in [2], potendo così utilizzare il relativo algoritmo di rilevamento deadlock (back-end) già esistente e verificato. Nella sezione 2 viene descritta la sintassi del bytecode Java direttamente nel formato *g4* di *ANTLR4*. Nella sezione 3 è illustrata la semantica operazionale e nella sezione 4 vengono riportati degli esempi di deadlock che si possono verificare tra due thread in esecuzione. Infine nella sezione 5 si presenta il sistema di tipi comportamentali.

2 La sintassi

In questa sezione viene riportata la grammatica nella sintassi *ANTLR4* che permette la creazione automatica del parser Java utile alla costruzione dell' *Abstract Sintax Tree* a partire da una sintassi ridotta del bytecode che tiene conto solo delle istruzioni che operano su oggetti e su tipi di dato intero.

```
grammar bytecodeConcurrency;

classfile
        : (classDec innerClass? constantPool '{' fields*
           methodDeclaration+ '}' innerClass?) EOF
        ;

classDec
        : classModifier? ('class'|'enum') packageAndClassName ('extends'
           packageAndClassName)? (('implements' packageAndClassName) (','
            packageAndClassName)*)?
```

```
        ;

classModifier
        : 'public'
        | 'static'
        | 'final'
        | 'abstract'
        ;

innerClass
        : 'InnerClasses:' (('static' ('= '|ref|'of')+ | ref) ';')+
        ;

fields
        : fieldModifier* type fieldName';'
        ;

fieldModifier
        : classModifier
        | 'protected'
        | 'private'
        ;

fieldName: packageAndClassName;

constantPool
        : 'Constant pool:' tableEntries
        ;

tableEntries: tableEntry+;


tableEntry
        : ref '= ' constantAndInfo
        ;



constantAndInfo
        : 'Class' ref
        | 'Fieldref' ref'.'ref
        | 'Methodref' ref'.'ref
        | 'InterfaceMethodref' ref'.'ref
        |  STRING ref
        | 'Integer' num
        | 'Float' DEC
        | 'Long' num'l'
        | 'Double' DEC
        | 'NameAndType' ref':'ref
        | 'Utf8' ((Identifier | identifierExtended) | NAT | '%' | ';' | '
            (' | ')'|'/' | '-' | '<' | '>' | ';' | '[' | ']' | '.' | ':' |
             '\"' | '!' | '?' | '^' |'\\'|',')*
        ;
```

```
methodDeclaration
        : methodModifier* (methodHeader|'{}') ';' methodBody
        ;

methodModifier
        : fieldModifier
        | 'abstract'
        | 'synchronized'
        | 'native'
        | 'strictfp'
        ;

methodHeader
        : result? methodDeclarator throws_?
        ;

result
        : { ! _input.LT(1).getText().substring(_input.LT(1).getText().
            length() - 1).equals("(")}? type
        | 'void'
        ;

type
        : primitiveType
        | referenceType
        ;


primitiveType
        : numericType
        ;


numericType
        : integralType
        ;


integralType
        | 'int'
        ;

referenceType
        : packageAndClassName
        ;

packageAndClassName
        : ((Identifier|identifierExtended) '['* ']'*)
        |   packageAndClassName '.' ((Identifier|identifierExtended) '['*
            ']'*)
        ;
```

```
methodDeclarator
        : methodName '(' formalParameters? ')'
        ;

methodName: packageAndClassName;

formalParameters
        : formalParameter (',' formalParameter)*
        ;

formalParameter
        : 'final'? (type |THIS)
        ;

throws_
        : 'throws' exceptionTypeList
        ;

exceptionTypeList
        : exceptionType (',' exceptionType)*
        ;

exceptionType
        : packageAndClassName
        ;

methodBody
        : 'Code:' instructions
        ;

instructions: instruction+;

instruction
        : INDEX ('aconst_null'
        | 'aload' NAT
        |  ALOAD
        | 'areturn'
        | 'astore' NAT
        | ASTORE
        | 'athrow'
        | 'dup'
        | 'getfield' ref
        | 'getstatic' ref
        | 'goto' NAT
        | 'iadd'
        | 'iconst_m1'
        | 'iconst_0'
        | 'iconst_1'
        | 'iconst_2'
        | 'iconst_3'
        | 'iconst_4'
        | 'iconst_5'
        | 'bipush' num
```

```
        | 'idiv'
        | 'if_acmpeq'
        | 'if_acmpne'
        | 'if_icmpeq' NAT
        | 'if_icmpne' NAT
        | 'if_icmplt' NAT
        | 'if_icmpge' NAT
        | 'if_icmpgt' NAT
        | 'if_icmple' NAT
        | 'ifeq' NAT
        | 'ifne' NAT
        | 'iflt' NAT
        | 'ifle' NAT
        | 'ifgt' NAT
        | 'ifge' NAT
        | 'ifnonnull'
        | 'ifnull'
        | 'iinc' NAT num
        | 'iload' NAT
        | ILOAD
        | 'imul'
        | 'invokespecial' ref
        | 'invokestatic' ref
        | 'invokevirtual' ref
        | 'irem'
        | 'ireturn'
        | 'istore' NAT
        | ISTORE
        | 'isub'
        | 'ldc' ref
        | 'ldc_w' ref
        | 'monitorenter'
        | 'monitorexit'
        | 'new' ref
        | 'pop'
        | 'putfield' ref
        | 'putstatic' ref
        | 'return')
        ;

INDEX
        : NAT':'
        ;

ref
        : REFNUM
        ;

ALOAD
        : 'aload_' NAT
        ;

ASTORE
```

```
217         : 'astore_'NAT
            ;

219
    ILOAD
221         : 'iload_'NAT
            ;

223
    ISTORE
225         : 'istore_'NAT
            ;

227
    REFNUM
229         : '#'[1-9][0-9]*
            ;

231
    num
233         : NAT
            | INT
235         ;

237 NAT
            : [0-9]+
239         ;


241
    INT
243         : '-'? NAT
            ;

245
    DEC
247         : '-'? [0-9]*'.'[0-9]*
            ;

249
    THIS
251         : 'this'
            ;

253
    STRING
255         : 'String'
            ;

257
    Identifier
259         : JavaLetter JavaLetterOrDigit*
            ;

261
    identifierExtended
263         : Identifier
            | STRING
265         | THIS
            ;

267
    fragment
269 JavaLetter
```

```
        : [a-zA-Z$_]
        | ~[\u0000-\u00FF\uD800-\uDBFF]{Character.isJavaIdentifierStart(
            _input.LA(-1))}?
        | [\uD800-\uDBFF] [\uDC00-\uDFFF]{Character.isJavaIdentifierStart
            (Character.toCodePoint((char)_input.LA(-2), (char)_input.LA
            (-1)))}?
        ;

fragment
JavaLetterOrDigit
        : [a-zA-Z0-9$_]
        | ~[\u0000-\u00FF\uD800-\uDBFF]{Character.isJavaIdentifierPart(
            _input.LA(-1))}?
        | [\uD800-\uDBFF] [\uDC00-\uDFFF]{Character.isJavaIdentifierPart(
            Character.toCodePoint((char)_input.LA(-2), (char)_input.LA(-1)
            ))}?
        ;

CLASSFILE: 'Classfile ' ~[\r\n]+ {skip(); System.out.println("skippato
    CLASSFILE");};

LAST: 'Last modified '  [0-9|a-z|A-Z|','|'\-'|';'|' ']+ ' size ' [0-9]+ '
     bytes'  '\r'? '\n' {skip(); System.out.println("skippato LAST");};

MD5: 'MD5 checksum ' [0-9|a-z]+  '\r'? '\n' {skip(); System.out.println("
    skippato MD5");};

COMPILED: 'Compiled from ' ['"'|a-z|A-Z|'.']+  '\r'? '\n'{skip(); System.
    out.println("skippato COMPILED");};

ENCLOSINGMETHOD: 'EnclosingMethod: ' ['#'|0-9|'.']+ {skip(); System.out.
    println("skippato ENCLOSINGMETHOD");};

SOURCE: 'SourceFile: ' ['"'|a-z|A-Z|'.']+  '\r'? '\n'{skip(); System.out.
    println("skippato SOURCE");};

MINOR: 'minor version: ' [0-9]+  '\r'? '\n'{skip(); System.out.println("
    skippato MINOR");};

MAJOR: 'major version: ' [0-9]+  '\r'? '\n'{skip(); System.out.println("
    skippato MAJOR");};

FLAGS: 'flags:' [ A-Z|'_'|',']*  '\r'? '\n'{skip(); System.out.println("
    skippato FLAGS");};

DESCRIPTOR: 'descriptor:'  ~[\r\n]* {skip(); System.out.println("skippato
     DESCRIPTOR");};

EXCEPTIONTABLE: 'Exception table:'  '\r'? '\n'{skip(); System.out.println
    ("skippato EXCEPTIONTABLE");};

FROMTO: 'from' ' '+ 'to' ' '+ 'target' ' '+ 'type' '\r'? '\n'{skip();
    System.out.println("skippato FROMTO");};
```

```
305
FROMTONUMBERS: NAT [' '|'/t']+ NAT [' '|'/t']+ NAT [' '|'/t']+ [a-z]+  '\
    r'? '\n'{skip(); System.out.println("skippato FROMTONUMBERS");};
307
LINENUMBERTABLE: 'LineNumberTable:'  '\r'? '\n' {skip(); System.out.
    println("skippato LINENUMBERTABLE");};
309
LINENUMBERTABLECONTENT: 'line ' [0-9]+ ': ' [0-9]+  '\r'? '\n'{skip();
    System.out.println("skippato LINENUMBERTABLECONTENT");};
311
LOCALVARIABLETABLE: 'LocalVariableTable:'  '\r'? '\n' {skip(); System.out
    .println("skippato LOCALVARIABLETABLE");};
313
LOCALVARIABLECONTENT: 'Start  Length  Slot  Name  Signature' '\r'? '\n'{
    skip(); System.out.println("skippato LOCALVARIABLETABLECONTENT");};
315
LOCALVARIABLELINE: [0-9]+ [' '|'/t']+ [0-9]+ [' '|'/t']+ [0-9]+ [' '|'/t'
    ]+  ~[\r\n]+ {skip(); System.out.println("skippato
    LOCALVARIABLETABLELINE");};
317
STACKMAPTABLE: 'StackMapTable: number_of_entries = ' [0-9]+ '\r'? '\n'{
    skip(); System.out.println("skippato STACKMAPTABLE");};
319
FRAMETYPE: 'frame_type = ' [0-9]+ [' '|'*'|'/'|A-Z|a-z|'_'|0-9]*  '\r'? '
    \n'{skip(); System.out.println("skippato FRAMETYPE");};
321
OFFSETDELTA: 'offset_delta = ' [0-9]+ [' '|'*'|'/'|A-Z|a-z]*  '\r'? '\n'{
    skip(); System.out.println("skippato OFFSETDELTA");};
323
LOCALS: 'locals = ' '[' [A-Z|a-z|','|'/'|' ']* ']' '\r'? '\n'{skip();
    System.out.println("skippato LOCALS");};
325
STACK: 'stack =' ~[\r\n]*  {skip(); System.out.println("skippato STACK");
    };
327
STACKLOCALARGSSIZE: 'stack=' ( |[0-9]+ ', locals='[0-9]+ ', args_size='
    [0-9]+)  '\r'? '\n'{skip(); System.out.println("skippato
    STACKLOCALARGSSIZE");};
329
EXCEPTION: 'Exceptions:'  '\r'? '\n'{skip(); System.out.println("skippato
     EXCEPTION");};
331
THROWS: 'throws ' [A-Z|a-z|'.']+  '\r'? '\n'{skip(); System.out.println("
    skippato THROWS");};
333
//INNER: 'InnerClasses:'  '\r'? '\n'{skip(); System.out.println("skippato
     INNERCLASS");};
335
//INNERCLASSESREF: REFNUM';' {skip(); System.out.println("skippato
    INNERCLASSREF");};
337
//STATICINNERCLASS: 'static #'[0-9]+ ~[\r\n]* {skip(); System.out.println
    ("skippato STATICINNERCLASS");};
```

```
339
WS : [ \r\n\t] -> skip // skip spaces, tabs, newlines
341    ;


343


345 COMMENT
        :   '/*' .*? '*/' {skip(); System.out.println("skippato COMMENT");}
347     ;

349 LINE_COMMENT
        :   '//' ~[\r\n]* {skip();}
351     ;
```

3 La semantica operazionale

In questa sezione, prendendo spunto da quanto fatto in [2], si riporta una introduzione generale alla semantica operazionale. A seguire vengono elencate la totalità delle regole semantiche in funzione delle istruzioni che è possibile trovare all'interno del bytecode generato all'atto della compilazione.

Partendo da un contesto formato da un insieme di classi P, da un'astrazione generalizzante *Cpool* per le singole constant pool presenti in ciascuna classe e da un'astrazione generalizzante *ExcTable* che racchiude tutte le tabelle delle eccezioni dei metodi delle classi, ogni istruzione esegue una transizione tra configurazioni. Ogni configurazione è della forma:

$$P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, opStack_1) \cdot s_1, z_1 \rangle \ , \ \cdots \ , \\ \langle (pc^{D.m}, lvar_n, opStack_n) \cdot s_n, z_n \rangle \ ; \ W$$

A sinistra del $\Vdash_H$, come anticipato in precedenza, abbiamo il contesto formato da P, *Cpool* ed *ExcTable*. Per quanto riguarda la *Cpool*, in particolare, useremo la notazione $Cpool^C$ per denotare l'insieme delle costanti della classe C. H, invece, rappresenta lo *Heap* dove sono contenuti gli oggetti creati dinamicamente e verrà descritto meglio in seguito. A destra del $\Vdash_H$, vi sono l'insieme di thread del programma presenti nella Java Virtual Machine in un dato istante. In particolare, nella suddetta configurazione, vi sono n thread attualmente attivi, ognuno dei quali è racchiuso tra parentesi angolari, ed un insieme di thread in attesa di riprendere l'esecuzione che si assume siano contenuti nell'insieme W dei thread che aspettano di essere risvegliati da un qualche evento. Entrando nello specifico ogni Thread contiene:

- Una pila LIFO s contenente una serie di record di attivazione (o *frames*), racchiusi tra parentesi tonde, e che a loro volta contengono le informazioni relative a ciascun metodo chiamato durante l'esecuzione.
- Un set z di oggetti di cui si detiene il *lock*.

In ogni frame a sua volta sono contenute le seguenti informazioni:

- Un *program counter pc* che punta all'indirizzo della prossima istruzione da eseguire. In particolare $pc^{C.m}$ punterà ad un'istruzione all'interno del metodo m della classe C. Per ottenere il nome di questa istruzione dall'indirizzo puntato dal program counter useremo la notazione $P[pc^{C.m}]$.
- Un insieme di variabili locali *lvar* tra cui sono contenuti anche gli argomenti passati come parametri al metodo. Inoltre con la notazione $lvar(this)$ è possibile ottenere il riferimento all'oggetto su cui il metodo è stato chiamato, oppure il riferimento alla classe se il metodo è statico.
- un *operand Stack* LIFO *opStack* in cui vengono inseriti i riferimenti ad oggetti ed i valori interi necessari per effettuare le operazioni locali al metodo.

Lo *Heap* in questa semantica contiene:

- Tutti gli oggetti allocati tramite le istruzioni *new*. Ognuno di questi oggetti sarà costituito da:
  - un insieme di campi d'istanza. Il valore di un campo a di un oggetto o sarà accessibile con la notazione $H(o).a$. Per aggiornarne il valore a v si userà invece la notazione $H(o).a \mapsto v$
  - la classe di appartenenza dell'oggetto. Ed esempio, per recuperare la classe di appartenenza di un oggetto o si userà la notazione $H(o).class$
  - un wait set che contiene i riferimenti ai thread che hanno invocato $wait()$ sull'oggetto.
  - un contatore l che tiene traccia del numero di lock che un thread ha acquisito sull'oggetto (solo un thread per volta può possedere il lock di un oggetto). Questo campo è accessibile ed aggiornabile con la stessa notazione usata per i normali campi dell'oggetto, cioè con $H(o).l$ e $H(o).l \mapsto n$ (le istruzioni *getstatic*, *putstatic*, $getfield$, $putfield$ non possono agire su questo campo). Se il valore del campo l dell'oggetto è uguale a 0 significa che nessun thread ne ha il lock.
- Un oggetto speciale per ogni classe del programma, il cui scopo è quello di rappresentare la classe stessa e di permettere di accedere ai relativi campi statici. Sarà quindi possibile, ad esempio, recuperare il valore di un campo statico a della classe C mediante la notazione $H(C).a$ e aggiornarne il valore con $H(C).a \mapsto v$. Inoltre, come per gli oggetti istanziati con le *new*, ad ognuno di tali oggetti speciali che identificano una classe è associato un campo l che permette di acquisire e rilasciare il lock della classe corrispondente. Ad esempio, per accedere al valore del campo l della classe C si userà la notazione $H(C).l$ e per rilasciare il lock di tale classe si userà la notazione $H(C).l \mapsto 0$

Inoltre viene utilizzata la seguente notazione:

- v per valori interi, o per i riferimenti agli oggetti, e per entrambi
- $H(o) \mapsto (\rho^C_\perp, C)$ per allocare lo spazio per l'oggetto o della classe C nello heap assegnando un valore indefinito ai rispettivi campi d'istanza ed inizializzando il suo campo *class* con la classe opportuna. Questa situazione si ha eseguendo un'istruzione *new*, i campi dell'oggetto rimangono indefiniti in quanto la creazione di un istanza di una classe non è completa finché l'oggetto non viene inizializzato mediante una chiamata *invokespecial* al costruttore.
- $H(o) \mapsto \rho^C_{args}$ per inizializzare i campi dell'oggetto o di classe C con gli argomenti $args$ passati all'opportuno costruttore. I rimanti campi non gestiti dal costruttore verranno inizializzati con 0 nel caso si tratti di interi e *null* nel caso si tratti di riferimenti. Questo assegnamento completa la creazione di un nuovo oggetto allocato precedentemente con la *new*.
- $Cpool^C(ref) = [\text{“}Costante\text{”}, \text{“}Valore\text{”}]$ per indicare che ad una determinato indirizzo ref della *Cpool* della classe C si trova una entry costante-valore. La costante di una entry della *Constant Pool* è una tra quelle definite nella *Java Virtual Machine Specification* ed identifica il tipo di informazione che sarà contenuta nella parte valore della entry.

– $ExcTable^{C.m}(pc^{C.m}, H(o).class) = L$ per indicare che la tabella delle eccezioni relativa al metodo C.m è in grado di gestire un'eccezione (che è un oggetto a tutti gli effetti) o di tipo $H(o).class$ che si verifica alla riga puntata da $pc^{C.m}$ mediante un $Exception\ Handler$ (una sequenza di istruzioni bytecode) che si trova all'indirizzo L (che deve essere per forza un'indirizzo interno al metodo $C.m$). Se invece tale $ExcTable^{C.m}$ non fosse in grado di gestire una tale eccezione, quando questa si verifica a quella riga, la notazione sarebbe la seguente: $ExcTable^{C.m}(pc^{C.m}, H(o).class) = \perp$

Per concludere, l'insieme dei *waiting set* W è definito come: $W = \{W_{o_1}, \ldots, W_{o_n}$ $, J_{o_1}, \ldots, J_{o_n}\}$. Ognuno dei W_{o_i} raggruppa l'insieme dei processi in attesa che un altro thread effettui una $notify$ sull'oggetto o_i. Mentre ognuno dei J_{o_i} raggruppa l'insieme dei thread in attesa della terminazione dell'oggetto o_i. Si assume quindi, in quest'ultimo caso, che o_i sia un thread, ovvero un oggetto istanza della classe $Thread$ su cui è possibile invocare $o_i.join()$ per attenderne la terminazione Per il singolo *waiting set* utilizziamo la notazione $W_{o_i} = \{\langle(pc_k, lvar_k, opStack_k) \cdot s_k, z_k\rangle_{n_k} |\ k \in K\}$ dove K rappresenta l'insieme dei Thread indicizzati ed in attesa della *notify* sull'oggetto o_i. Mentre per il singolo *waiting set* J_{o_i} utilizziamo la notazione $J_{o_i} = \{\langle(pc_k, lvar_k, opStack_k) \cdot s_k, z_k\rangle_{o_i} |\ k \in K\}$ dove K rappresenta l'insieme dei thread indicizzati ed in attesa della terminazione dell'oggetto o_i che contiene il metodo *run()* del thread corrispondente. Infine abbiamo:

– $\emptyset^w_{o_i}$ che indica che non c'è nessun thread in attesa di una *notify* sull'oggetto o_i.
– $\emptyset^j_{o_i}$ che indica che non c'è nessun thread in attesa della terminazione dell'oggetto/thread o_i

Di seguito sono riportate le regole semantiche di riduzione. Ogni regola riporta solamente le componenti che partecipano alla riscrittura anche se ovviamente tale riscrittura si applica ad ogni configurazione che contiene tali componenti.

Partiamo dalle regole più semplici: *pop*, *iinc*, *iload*, *aload*, *istore*, *astore*, *goto*, etc...

$$\frac{P[pc^{C.m}] = \texttt{pop}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, e \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iinc}\ \ index\ v}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar[index \mapsto lvar(index) + v], opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iload}\ \ index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, lvar(index) \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iload\_}index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, lvar(index) \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{aload}\ \ index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, lvar(index) \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{aload\_}index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, lvar(index) \cdot opStack \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{istore}\ \ index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar[index \mapsto v], opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{istore\_}index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar[index \mapsto v], opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{astore}\ \ index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar[index \mapsto o], opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{astore\_}index}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar[index \mapsto o], opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{goto}\ L^{C.m}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iadd}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, v'' + v' \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{isub}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, v'' - v' \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{imul}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, v'' \times v' \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{idiv} \quad v' \neq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, v''/v' \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{irem} \quad v' \neq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, v''\%v' \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iconst\_m1}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, -1 \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iconst\_}n \quad 0 \leq n \leq 5}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, n \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{aconst\_null}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar,\ opStack) \cdot s, z)\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar,\ null \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{bipush}\ \ v \quad -128 \leq v \leq 127}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, v \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{dup}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, e \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, e \cdot e \cdot opStack) \cdot s, z\rangle}$$

Vediamo adesso le istruzioni di salto condizionale:

$$\frac{P[pc^{C.m}] = \texttt{if\_acmpeq}\ L^{C.m} \qquad o' = o''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o' \cdot o'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_acmpeq}\ L^{C.m} \qquad o' \neq o''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o' \cdot o'' \cdot opStack) \cdot s, z)\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_acmpne}\ L^{C.m} \qquad o' \neq o''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o' \cdot o'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_acmpne}\ L^{C.m} \qquad o' = o''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o' \cdot o'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpeq}\ L^{C.m} \qquad v' = v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpeq}\ L^{C.m} \qquad v' \neq v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpne}\ L^{C.m} \qquad v' \neq v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpne}\ L^{C.m} \qquad v' = v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v' \cdot v'' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmple}\ L^{C.m} \qquad v' > v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmple}\ L^{C.m} \qquad v' \leq v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmplt}\ L^{C.m} \qquad v' \geq v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmplt}\ L^{C.m} \qquad v' < v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpge}\ L^{C.m} \qquad v' < v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpge}\ L^{C.m} \qquad v' \geq v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpgt}\ L^{C.m} \qquad v' \leq v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{if\_icmpgt}\ L^{C.m} \qquad v' > v''}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v'' \cdot v' \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifeq}\ L^{C.m}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, 0 \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifeq}\ L^{C.m} \quad v \neq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifne}\ L^{C.m}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, 0 \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifne}\ L^{C.m} \quad v \neq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iflt}\ L^{C.m} \qquad v \geq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{iflt}\ L^{C.m} \qquad v < 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifle}\ L^{C.m} \qquad v > 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack, \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifle}\ L^{C.m} \qquad v \leq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifgt}\ L^{C.m} \qquad v \leq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifgt}\ L^{C.m} \qquad v > 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifge}\ L^{C.m} \qquad v < 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifge}\ L^{C.m} \qquad v \geq 0}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, v \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifnull}\ L^{C.m}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, null \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifnull}\ L^{C.m} \quad o \neq null}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifnonnull}\ L^{C.m}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, null \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc + 1^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ifnonnull}\ L^{C.m} \quad o \neq null}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (L^{C.m}, lvar, opStack) \cdot s, z\rangle}$$

Consideriamo adesso le regole formali per le *invokespecial*, *invokestatic* e le *invokevirtual*. Le prime vengono utilizzate per invocare un costrutture, le seconde per invocare metodi statici e le terze per tutti gli altri metodi.

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{invokespecial}\ \ ref \\ Cpool^C(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\ Cpool^C(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\ Cpool^C(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\ Cpool^C(ref_3) = [\text{“}Utf8\text{”}, \text{“} < init > \text{”}] \\ Cpool^C(ref_4) = [\text{“}Utf8\text{”}, \text{“}(T_{par_1}, \ldots, T_{par_n})V\text{”}] \\ H(o) = (\rho^D_\perp, D) \\ H' = H[o \mapsto \rho^D_{args}] \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, arg_n \cdot \ldots \cdot arg_1 \cdot o \cdot opStack) \cdot s, z\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle \end{array}}$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokestatic}\ \ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}foo\text{”}] \\
Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}(T_{par_1}, \ldots, T_{par_n})T_{res}\text{”}] \\
synchronized \notin modifier(D.foo)
\end{gathered}
$$

$$
\begin{gathered}
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, arg_n \cdot, \ldots, \cdot arg_1 \cdot opStack_1) \cdot s, z\rangle \rightarrow \\
\Vdash_H \langle (1^{D.foo}, lvar_2[0 \mapsto arg_1, \ldots, n-1 \mapsto arg_n, this \mapsto D], \epsilon) \cdot \\
(pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z\rangle
\end{gathered}
$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokestatic}\ \ ref \\
Cpool^C(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^C(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\
Cpool^C(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^C(ref_3) = [\text{“}Utf8\text{”}, \text{“}foo\text{”}] \\
Cpool^C(ref_4) = [\text{“}Utf8\text{”}, \text{“}(T_{par_1}, \ldots, T_{par_n})T_{res}\text{”}] \\
synchronized \in modifier(D.foo) \\
H(D).l = n,\ n > 0 \\
H' = H(D).l \mapsto n + 1
\end{gathered}
$$

$$
\begin{gathered}
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, arg_n \cdot \ldots \cdot arg_1 \cdot opStack_1) \cdot s, z \cup \{D\}\rangle \rightarrow \\
\Vdash_{H'} \langle (1^{D.foo}, lvar_2[0 \mapsto arg_1, \ldots, n-1 \mapsto arg_n, this \mapsto D], \epsilon) \cdot \\
(pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z \cup \{D\}\rangle
\end{gathered}
$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokestatic}\ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}foo\text{”}] \\
Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}(T_{par_1}, \ldots, T_{par_n})T_{res}\text{”}] \\
syncronized \in modifier(D.foo) \\
H(D).l = 0 \\
H' = H(D).l \mapsto 1
\end{gathered}
$$

$$
\begin{gathered}
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, arg_n \cdot \ldots \cdot arg_1 \cdot opStack_1) \cdot s, z \backslash \{D\}\rangle \rightarrow \\
\Vdash_{H'} \langle (1^{D.foo}, lvar_2[0 \mapsto arg_1, \ldots, n-1 \mapsto arg_n, this \mapsto D], \epsilon) \cdot \\
(pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z \uplus \{D\}\rangle
\end{gathered}
$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokevirtual}\ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}foo\text{”}] \\
Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}(T_{par_1}, \ldots, T_{par_n})T_{res}\text{”}] \\
synchronized \notin modifier(D.foo)
\end{gathered}
$$

$$
\begin{gathered}
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, arg_n \cdot \ldots \cdot arg_1 \cdot o \cdot opStack_1) \cdot s, z \rangle \rightarrow \\
\Vdash_H \langle (1^{D.foo}, lvar_2[0 \mapsto o, 1 \mapsto arg_1, \ldots, n \mapsto arg_n, this \mapsto o], \epsilon) \cdot \\
(pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z \rangle
\end{gathered}
$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokevirtual}\ \ ref \\
Cpool^C(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^C(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\
Cpool^C(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^C(ref_3) = [\text{“}Utf8\text{”}, \text{“}foo\text{”}] \\
Cpool^C(ref_4) = [\text{“}Utf8\text{”}, \text{“}(T_{par_1}, \ldots, T_{par_n})V\text{”}] \\
synchronized \in modifier(D.foo) \\
H(o).l = n,\ n > 0 \\
H' = H(o).l \mapsto n + 1
\end{gathered}
$$

$$
\begin{gathered}
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, arg_n \cdot \ldots \cdot arg_1 \cdot o \cdot opStack_1) \cdot s, z \cup \{o\} \rangle \rightarrow \\
\Vdash_{H'} \langle (1^{D.foo}, lvar_2[0 \mapsto o, 1 \mapsto arg_1, \ldots, n \mapsto arg_n, this \mapsto o], \epsilon) \cdot \\
(pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z \cup \{o\} \rangle
\end{gathered}
$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokevirtual}\ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}D\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}foo\text{”}] \\
Cpool^c(ref_4) = [\text{“”}Utf8\text{”}, \text{“}(Tpar_1, \ldots, Tpar_n)Tres\text{”}] \\
syncronized \in modifier(D.foo) \\
H(o).l = 0 \\
H^i = H(o).l \mapsto 1
\end{gathered}
$$

$$
\begin{gathered}
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, arg_n \cdot \ldots \cdot arg_1 \cdot o \cdot opStack_1) \cdot s, z \backslash \{o\} \rangle \rightarrow \\
\Vdash_H \langle (1^{D.foo}, lvar_2[0 \mapsto o, 1 \mapsto arg_1, \ldots, n \mapsto arg_n, this \mapsto o], \epsilon) \cdot \\
(pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z \uplus \{o\} \rangle
\end{gathered}
$$

La regola seguente gestisce la creazione di un nuovo thread in seguito all'invocazione del metodo *start*.

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{invokevirtual}\ \ ref \\ o\ instance\ of\ java/lang/Thread \\ Cpool^C(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\ Cpool^C(ref_1) = [\text{“}Class\text{”}, \text{”}java/lang/Thread\text{”}] \\ Cpool^C(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\ Cpool^C(ref_3) = [\text{“}Utf8\text{”}, \text{“}start\text{”}] \\ Cpool^C(ref_4) = [\text{“}Utf8\text{”}, \text{“}()V\text{”}]\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, o \cdot opStack_1) \cdot s, z\rangle \rightarrow \\ \Vdash_H \langle (pc^{C.m} + 1, lvar_1, opStack_1) \cdot s, z\rangle\ , \\ \langle (1^{java/lang/Thread.run}, lvar_2[0 \mapsto o, this \mapsto o], \epsilon), \epsilon\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{invokevirtual}\ \ ref \\ Cpool^C(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\ Cpool^C(ref_1) = [\text{“}Class\text{”}, \text{“}java/lang/Thread\text{”}] \\ Cpool^C(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\ Cpool^C(ref_3) = [\text{“}Utf8\text{”}, \text{“}join\text{”}] \\ Cpool^C(ref_4) = [\text{“}Utf8\text{”}, \text{“}()V\text{”}] \\ o = lvar_1^{t2}(this)\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, o \cdot opStack_1) \cdot s_{t_1}, z_{t_1}\rangle\ , \\ \langle s_{t2} \cdot (pc^{java/lang/Thread.run}, lvar_1^{t2}, opStack_1^{t2}), z_{t_2}\rangle; J_o \rightarrow \\ \Vdash_H \langle s_{t2} \cdot (pc^{java/lang/Thread.run}, lvar_1^{t2}, opStack_1^{t2}), z_{t_2}\rangle; \\ J_o \uplus \langle (pc^{C.m} + 1, lvar_1, opStack_1) \cdot s_{t_1}, z_{t_1}\rangle_o\end{array}}$$

La seguente regola è relativa alla ripartenza dei thread che si erano precedentemente messi in attesa della terminazione di uno stesso thread (oggetto o) mediante la chiamata a $join()$. Appena tale thread termina tutti quelli che erano in attesa ripartono.

$$\frac{\begin{array}{c}P[pc^{java/lang/Thread.run}] = \texttt{return} \\ o = lvar(this)\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{java/lang/Thread.run}, lvar, opStack), z_1\rangle; \\ \emptyset_o^j \uplus (\langle (pc_k, lvar_k, opStack_k) \cdot s_k, z_k)\rangle_o)^{k \in K} \rightarrow \\ \Vdash_H (\langle (pc_k, lvar_k, opStack_k) \cdot s_k, z_k)\rangle)^{k \in K}; \emptyset_o^j\end{array}}$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokevirtual}\ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}java/lang/Object\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}notify\text{”}] \\
Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}()V\text{”}] \\
H(o).l = n,\ n > 0 \\
\hline
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, o \cdot opStack_1) \cdot s_1, z_1 \cup \{o\} \rangle; \\
W_o \uplus \{\langle (pc^{D.m}, lvar_2, opStack_2) \cdot s_2, z_2 \rangle_n\} \ \rightarrow \\
\Vdash_H \langle (pc^{C.m} + 1, lvar_1, opStack_1) \cdot s_1, z_1 \cup \{o\} \rangle\ ,\ \langle (pc^{D.m}, lvar_2, opStack_2) \cdot s_2, z_2 \rangle_n\ ; W_o
\end{gathered}
$$

Nella regola seguente il wait set dell'oggetto o è vuoto e pertanto nessuna notifica viene fatta

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokevirtual}\ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}java/lang/Object\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}notify\text{”}] \\
Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}()V\text{”}] \\
H(o).l = n,\ n > 0 \\
\hline
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \cup \{o\} \rangle; \emptyset_o^w \rightarrow \\
\Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z \cup \{o\} \rangle; \emptyset_o^w
\end{gathered}
$$

$$
\begin{gathered}
P[pc^{C.m}] = \texttt{invokevirtual}\ ref \\
Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\
Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}java/lang/Object\text{”}] \\
Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_3 : ref_4] \\
Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}notifyAll\text{”}] \\
Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}()V\text{”}] \\
H(o).l = n,\ n > 0 \\
W_o = \{\langle (pc_k, lvar_k, opStack_k) \cdot s_k, z_k) \rangle_{n_k} \mid k \in K\}_o \\
\hline
P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \cup \{o\} \rangle; W_o \rightarrow \\
\Vdash_H \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z \cup \{o\} \rangle, (\langle (pc_k, lvar_k, opStack_k) \cdot s_k, z_k) \rangle_{n_k})^{k \in K}; \emptyset_o^w
\end{gathered}
$$

$$\frac{\begin{array}{c} P[pc_{t1}^{C.m}] = \texttt{invokevirtual}\ ref \\ Cpool^c(ref) = [\text{“}Methodref\text{”}, ref_1.ref_2] \\ Cpool^c(ref_1) = [\text{“}Class\text{”}, \text{“}java/lang/Object\text{”}] \\ Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref3 : ref4] \\ Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}wait\text{”}] \\ Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}()V\text{”}] \\ H(o).l = n,\ n > 0 \\ H' = H(o).l \mapsto 0 \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \cup \{o\} \rangle; W_o \rightarrow \\ \Vdash_{H'}; W_o \uplus \langle (pc^{C.m}, lvar, opStack) \cdot s, z \cup \{o\} \rangle_n \end{array}}$$

Un thread che viene risvegliato da una *notify* (o *notifyAll*) può continuare la sua esecuzione solamente quando il lock dell'oggetto in cima al suo stack (quello su cui aveva invocato la *wait*) viene rilasciato. In questo caso il thread può riacquisire il lock di tale oggetto settandone il relativo campo l al valore n in modo da ripristinare il numero di lock acquisiti prima della chiamata *wait*.

$$\frac{\begin{array}{c} H(o).l = 0 \\ H' = H(o).l \mapsto n \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \rangle_n \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z \rangle \end{array}}$$

Consideriamo adesso le varie *return* che siano esse utilizzate per ritornare un tipo *void*, un intero oppure un oggetto.

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{return} \\ synchronized \notin modifier(C.m)\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z\rangle \rightarrow \\ \Vdash_H \langle pc^{D.m}, lvar_2, opStack_2) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{return} \\ synchronized \in modifier(C.m) \\ o = lvar_1(this) \\ H(o).l = 1 \\ H' = H(o).l \mapsto 0\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z \uplus \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, opStack_2) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{return} \\ synchronized \in modifier(C.m) \\ o = lvar_1(this) \\ H(o).l = n, n > 1 \\ H' = H(o).l \mapsto n - 1\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, opStack_1) \cdot (pc^{D.m}, \langle lvar_2, opStack_2) \cdot s, z \cup \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, opStack_2 \cdot s, z \cup \{o\}\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{ireturn} \\ synchronized \notin modifier(C.m)\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, v \cdot opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, v \cdot opStack_2) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{ireturn} \\ synchronized \in modifier(C.m) \\ o = lvar_1(this) \\ H(o).l = n, n > 1 \\ H' = H(o).l \mapsto n - 1\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, v \cdot opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z \cup \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, v \cdot opStack_2) \cdot s, z \cup \{o\}\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{ireturn} \\ synchronized \in modifier(C.m) \\ o = lvar_1(this) \\ H(o).l = 1 \\ H' = H(o).l \mapsto 0\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, v \cdot opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z \uplus \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, v \cdot opStack_2) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{areturn} \\ synchronized \notin modifier(C.m)\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, o \cdot opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z\rangle \rightarrow \\ \Vdash_H \langle (pc^{D.m}, lvar_2, o \cdot opStack_2) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{areturn} \\ synchronized \in modifier(C.m) \\ o' = lvar_1(this) \\ H(o').l = 1 \\ H' = H(o').l \mapsto 0\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, o \cdot opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z \uplus \{o'\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, o \cdot opStack_2) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{areturn} \\ synchronized \in modifier(C.m) \\ o' = lvar_1(this) \\ H(o').l = n, n > 1 \\ H' = H(o').l \mapsto n - 1\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar_1, o \cdot opStack_1) \cdot (pc^{D.m}, lvar_2, opStack_2) \cdot s, z \cup \{o'\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{D.m}, lvar_2, o \cdot opStack_2) \cdot s, z \cup \{o'\}\rangle\end{array}}$$

Consideriamo adesso *putstatic*, *getstatic*, *putfield* e *getfield* utilizzate per settare e recuperare un valore rispettivamente da un campi statici o d'istanza.

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{putstatic}\ \ ref \\ Cpool^C(ref) = [\text{“}Fieldref\text{”}, ref_1.ref_2] \\ Cpool^C(ref_1) = [\text{“}Class\text{”}, ref_3] \\ Cpool^C(ref_3) = [\text{“}Utf8\text{”}, \text{“}D\text{”}] \\ Cpool^C(ref_2) = [\text{“}NameAndType\text{”}, ref_4 : ref_5] \\ Cpool^C(ref_4) = [\text{“}Utf8\text{”}, \text{“}a\text{”}] \\ Cpool^C(ref_5) = [\text{“}Utf8\text{”}, \text{“}T\text{”}] \\ H' = H(D).a \mapsto e \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, e \cdot opStack) \cdot s, z\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z) \end{array}}$$

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{getstatic}\ ref \\ Cpool^c(ref) = [\text{“}Fieldref\text{”}, ref_1.ref_2] \\ Cpool^c(ref_1) = [\text{“}Class\text{”}, ref_3] \\ Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}D\text{”}] \\ Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_4 : ref_5] \\ Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}a\text{”}] \\ Cpool^c(ref_5) = [\text{“}Utf8\text{”}, \text{“}T\text{”}] \\ H(D).a = e \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \\ \Vdash_H \langle (pc + 1^{C.m}, lvar, e \cdot opStack) \cdot s, z\rangle \end{array}}$$

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{putfield}\ ref \\ Cpool^c(ref) = [\text{“}Fieldref\text{”}, ref_1.ref_2] \\ Cpool^c(ref_1) = [\text{“}Class\text{”}, ref_3] \\ Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}D\text{”}] \\ Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_4 : ref_5] \\ Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}a\text{”}] \\ Cpool^c(ref_5) = [\text{“}Utf8\text{”}, \text{“}T\text{”}] \\ H' = H(o).a \mapsto e \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, e \cdot o \cdot opStack) \cdot s, z \rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z \rangle \end{array}}$$

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{getfield}\ ref \\ Cpool^c(ref) = [\text{“}Fieldref\text{”}, ref_1.ref_2] \\ Cpool^c(ref_1) = [\text{“}Class\text{”}, ref_3] \\ Cpool^c(ref_3) = [\text{“}Utf8\text{”}, \text{“}D\text{”}] \\ Cpool^c(ref_2) = [\text{“}NameAndType\text{”}, ref_4 : ref_5] \\ Cpool^c(ref_4) = [\text{“}Utf8\text{”}, \text{“}a\text{”}] \\ Cpool^c(ref_5) = [\text{“}Utf8\text{”}, \text{“}T\text{”}] \\ H(o).a = e \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \rangle \rightarrow \\ \Vdash_H \langle (pc^{C.m} + 1, lvar, e \cdot opStack) \cdot s, z \rangle \end{array}}$$

Una delle istruzioni più importanti da considerare è quella relativa alla creazione di nuovi oggetti mediante l'operatore *new*.

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{new}\ \ ref \\ Cpool^c(ref) = [\text{“}Class\text{”}, ref_1] \\ Cpool^c(ref_1) = [\text{“}Utf8\text{”}, \text{“}D\text{”}] \\ o \notin dom(H) \\ H' = H[o \mapsto (\rho^D_\perp, D)] \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, o \cdot opStack) \cdot s, z\rangle \end{array}}$$

Di seguito si riportano, invece, le importanti operazioni di presa e rilascio del lock tramite *monitorenter* e *monitorexit*.

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{monitorexit} \\ H(o).l = 1 \\ H' = H(o).l \mapsto 0 \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \uplus \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z\rangle \end{array}}$$

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{monitorexit} \\ H(o).l = n,\ \ n > 1 \\ H' = H(o).l \mapsto n - 1 \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \cup \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z \cup \{o\}\rangle \end{array}}$$

$$\frac{\begin{array}{c} P[pc^{C.m}] = \texttt{monitorenter} \\ H = H(o).l = 0 \\ H' = H(o).l \rightarrow\ 1 \end{array}}{\begin{array}{c} P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \backslash \{o\}\rangle \rightarrow \\ \Vdash_{H'} \langle (pc^{C.m} + 1, lvar, opStack) \cdot s, z \uplus \{o\}\rangle \end{array}}$$

$$\frac{P[pc^{C.m}] = \texttt{monitorenter} \quad H = H(o).l = n,\ n > 0 \quad H' = H(o).l \rightarrow\ n+1}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z \cup \{o\}\rangle \rightarrow \Vdash_{H'} \langle (pc^{C.m}+1, lvar, opStack) \cdot s, z \cup \{o\}\rangle}$$

Consideriamo adesso le istruzioni per recuperare una costante dalla *Constant Pool* ed inserirla sull' *operand Stack*[1].

$$\frac{P[pc^{C.m}] = \texttt{ldc}\ \ ref \quad Cpool^C(ref) = [\text{“}Integer\text{”}, v]}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m}+1, lvar, v \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ldc}\ \ ref \quad Cpool^C(ref) = [\text{“}String\text{”}, ref_1] \quad Cpool^C(ref_1) = [\text{“}Utf8\text{”}, \text{“}text\text{”}] \quad String\ o = \text{“}text\text{”}}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m}+1, lvar, o \cdot opStack) \cdot s, z\rangle}$$

$$\frac{P[pc^{C.m}] = \texttt{ldc}\ \ ref \quad Cpool^C(ref) = [\text{“}Class\text{”}, ref_1] \quad Cpool^C(ref_1) = [\text{“}Utf8\text{”}, \text{“}D\text{”}] \quad o = D}{P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, opStack) \cdot s, z\rangle \rightarrow \Vdash_H \langle (pc^{C.m}+1, lvar, o \cdot opStack) \cdot s, z\rangle}$$

[1] *ldc_w* ha le stesse regole semantiche di *ldc* ma può puntare ad indirizzi di memoria più estesi all'interno della *Constant pool*.

Di seguito le istruzioni inerenti al lancio di un'eccezione

$$\frac{\begin{array}{c}P[pc^{C.m}] = \texttt{athrow} \\ synchronized \notin modifier(C.m) \\ ExcTable^{C.m}(pc^{C.m}, H(o).class) = L\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C.m}, lvar, o \cdot opStack) \cdot s, z\rangle \rightarrow \\ \Vdash_H \langle (L, lvar, o) \cdot s, z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C_n.m_n}] = \texttt{athrow} \\ synchronized \notin modifier(C_n.m_n) \\ ExcTable^{C_n.m_n}(pc^{C_n.m_n}, H(o).class) = \bot \\ ExcTable^{C_i.m_i}(pc^{C_i.m_i}, H(o).class) = L\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C_n.m_n}, lvar_n, o \cdot opStack_n) \cdot \ldots \cdot \\ (pc^{C_i.m_i}, lvar_i, opStack_i) \cdot \ldots \cdot (pc^{C_1.m_1}, lvar_1, opStack_1), z\rangle \rightarrow \\ \Vdash_H \langle (L, lvar_i, o) \cdot \ldots \cdot (pc^{C_1.m_1}, lvar_1, opStack_1), z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C_n.m_n}] = \texttt{athrow} \\ synchronized \in modifier(C_n.m_n) \\ ExcTable^{C_n.m_n}(pc^{C_n.m_n}, H(o).class) = \bot \\ ExcTable^{C_i.m_i}(pc^{C_i.m_i}, H(o).class) = L \\ o' = lvar_n(this) \\ H(o').l = 1 \\ H' = H(o').l \mapsto 0\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C_n.m_n}, lvar_n, o \cdot opStack_n) \cdot \ldots \cdot \\ (pc^{C_i.m_i}, lvar_i, opStack_i) \cdot \ldots \cdot (pc^{C_1.m_1}, lvar_1, opStack_1), z \uplus \{o'\}\rangle \rightarrow \\ \Vdash_{H'} \langle (L, lvar_i, o) \cdot \ldots \cdot (pc^{C_1.m_1}, lvar_1, opStack_1), z\rangle\end{array}}$$

$$\frac{\begin{array}{c}P[pc^{C_n.m_n}] = \texttt{athrow} \\ synchronized \in modifier(C_n.m_n) \\ ExcTable^{C_n.m_n}(pc^{C_n.m_n}, H(o).class) = \bot \\ ExcTable^{C_i.m_i}(pc^{C_i.m_i}, H(o).class) = L \\ o' = lvar_n(this) \\ H(o').l = n, n > 1 \\ H' = H(o').l \mapsto n - 1\end{array}}{\begin{array}{c}P, Cpool, ExcTable \Vdash_H \langle (pc^{C_n.m_n}, lvar_n, o \cdot opStack_n) \cdot \ldots \cdot \\ (pc^{C_i.m_i}, lvar_i, opStack_i) \cdot \ldots \cdot (pc^{C_1.m_1}, lvar_1, opStack_1), z \cup \{o'\}\rangle \rightarrow \\ \Vdash_{H'} \langle (L, lvar_i, o) \cdot \ldots \cdot (pc^{C_1.m_1}, lvar_1, opStack_1), z \cup \{o'\}\rangle\end{array}}$$

4 Esempi di deadlock tra due thread in esecuzione parallela

(1)

$$P[pc^{T1.m}] = monitorenter$$
$$P[pc^{T2.m}] = monitorenter$$

$$\langle (pc^{T1.m}, lvar_1, o \cdot opstack_1) \cdot s_1,\ z_1 \cup \{o'\}\ and\ z_1 \backslash \{o\}\ \rangle,$$
$$\langle (pc^{T2.m}, lvar_2, o' \cdot opstack_2) \cdot s_2,\ z_2 \cup \{o\}\ and\ z_2 \backslash \{o'\} \rangle$$

(2) In questa regola si guarda il caso con *invokevirtual* ma lo stesso vale per *invokestatic* se si considerano i lock sulle classi.

$$P[pc^{T1.m}] = invokevirtual\ o'.M\ (synchronized \in modifier(o'.M))$$
$$P[pc^{T2.m}] = invokevirtual\ o.M\ (synchronized \in modifier(o.M))$$

$$\langle (pc^{T1.m}, lvar_1, arg_n \cdot \ldots \cdot arg_1 \cdot o \cdot opstack_1) \cdot s_1,\ z_1 \cup \{o'\}\ and\ z_1 \backslash \{o\}\ \rangle,$$
$$\langle (pc^{T2.m}, lvar_2, arg_n \cdot \ldots \cdot arg_1 \cdot o' \cdot opstack_2) \cdot s_2,\ z_2 \cup \{o\}\ and\ z_2 \backslash \{o'\} \rangle$$

(3)

$$P[pc^{T1.m}] = invokevirtual\ o'.wait$$
$$P[pc^{T2.m}] = invokevirtual\ o.join$$
$$lvar_1(this) = o$$

$$; W_{o'} \uplus \langle (pc^{T1.m}, lvar_1, opStack_1), z_1 \rangle_n$$
$$J_o \uplus \langle (pc^{T2.m}, lvar_2, opStack_2) \cdot s_2, z_2) \rangle_o$$

(4)

$$PC[pc^{T1.m}] = invokevirtual\ o'.wait$$
$$PC[pc^{T2.m}] = monitorenter$$

$$\langle (pc^{T2.m}, lvar_2, o'' \cdot opstack_2) \cdot s_2,\ z_2 \backslash \{o''\} \rangle;$$
$$W_{o'} \uplus \langle (pc^{T1.m}, lvar_1, opStack_1), z_1 \cup \{o''\} \rangle_n$$

(5)

$$P[pc^{T1.m}] = invokevirtual\ o'.wait$$
$$P[pc^{T2.m}] = invokevirtual\ o.M\ (synchronized \in modifier(o.M))$$

$$\langle (pc^{T2.m}, lvar_2, arg_n \cdot \ldots \cdot arg_1 \cdot o \cdot opstack_2) \cdot s_2, z_2 \backslash \{o\} \rangle;$$
$$W_{o'} \uplus \langle (pc^{T1.m}, lvar_1, opStack_1), z_1 \cup \{o\} \rangle_n$$

(6)

$$P[pc^{T1.m}] = invokevirtual\ o.join$$
$$P[pc^{T2.m}] = invokevirtual\ o'.join$$
$$lvar_1(this) = o'$$
$$lvar_2(this) = o$$

$$; J_o \uplus \langle (pc^{T1.m}, lvar_1, opStack_1) \cdot s_1, z_1) \rangle_o$$
$$J_{o'} \uplus \langle (pc^{T2.m}, lvar_2, opStack_2) \cdot s_2, z_2) \rangle_{o'}$$

(7)

$$P[pc^{T1.m}] = monitorenter$$
$$P[pc^{T2.m}] = invokevirtual\ o'.M\ (synchronized \in modifier(o'.M))$$

$$\langle (pc^{T1.m}, lvar_1, o \cdot opstack_1) \cdot s_1,\ z_1 \cup \{o'\}\ and\ z_1 \backslash \{o\}\ \rangle,$$
$$\langle (pc^{T2.m}, lvar_2, arg_n \cdot \ldots \cdot arg_1 \cdot o' \cdot opstack_2) \cdot s_2,\ z_2 \cup \{o\}\ and\ z_2 \backslash \{o'\} \rangle$$

(8)

$$P[pc^{T1.m}] = invokevirtual\ o.join$$
$$P[pc^{T2.m}] = monitorenter$$
$$lvar_2(this) = o$$

$$\langle (pc^{T2.m}, lvar_2, o' \cdot opstack_2) \cdot s_2,\ z_2 \backslash \{o'\} \rangle;$$
$$J_o \uplus \langle (pc^{T1.m}, lvar_1, opStack_1) \cdot s_1, z_1 \cup \{o'\}) \rangle_o$$

(9)

$$P[pc^{T1.m}] = invokevirtual\ o.join$$
$$P[pc^{T2.m}] = invokevirtual\ o'.M\ (synchronized \in modifier(o'.M))$$
$$lvar_2(this) = o$$

$$\langle (pc^{T2.m}, lvar_2, arg_n \cdot \ldots \cdot arg_1 \cdot o' \cdot opstack_2) \cdot s_2,\ z_2 \backslash \{o'\} \rangle;$$
$$J_o \uplus \langle (pc^{T1.m}, lvar_1, opStack_1) \cdot s_1, z_1 \cup \{o'\}) \rangle_o$$

5 Sistema di tipi comportamentali

In questa sezione introduciamo il sistema di tipi comportamentali utile alla definizione delle LAM. Il linguaggio delle LAM costituisce una rappresentazione astratta di un programma, già utilizzata in [2], che raccoglie in sè le informazioni minimali ed indispensabili all'individuazione delle dipendenze funzionali e quindi dei possibili deadlock.
Riconducendoci direttamente al linguaggio delle LAM, inoltre, è possibile applicare in maniera diretta l'algoritmo decisionale studiato ed implementato in [2].
Prima di addentrarci direttamente nella definizione delle regole di tipaggio, effettuiamo alcune considerazioni in merito alla notazione utilizzata.
Sia P il nostro programma in cui sono in esecuzione un certo numero di thread e sia P^σ la sequenza di istruzioni eseguite da un singolo thread. Come abbiamo discusso in precedenza ogni P^σ è costituito da uno stack di chiamate di metodo $m_n \cdot \ldots \cdot m_1$ dove m_n è l'ultima invocazione effettuata ed m_1 è il metodo $run()$ alla base dell'esecuzione del thread. Ognuno di questi metodi ha associato un insieme di variabili locali F^σ ed un operand stack S^σ.
Ogni tipo di oggetto nel programma
Abbiamo associato ad ogni istruzione bytecode esaminata precedentemente una regola di tipaggio che ci permette di associare a tale istruzione il relativo comportamento. In tal modo possiamo utilizzare la sequenza dei comportamenti delle singole istruzioni per ottenere la LAM di ogni P^σ che, una volta combinate ci permettono di ottenere la LAM di tutto il programma P. In particolare, per indicare che da un contesto $F^\sigma, S^\sigma, Z^\sigma, T^\sigma$ deriviamo la LAM L di un P^σ, usiamo la seguente notazione:

$$F^\sigma, S^\sigma, Z^\sigma, T^\sigma \vdash P^\sigma : L \ .$$

dove:

- $F^\sigma[i]$ è una mappa che, data un'istruzione i, restituisce l'array di tipi presenti nelle variabili locali del metodo al momento dell'esecuzione di tale istruzione.
- $S^\sigma[i]$ è una sequenza di tipi presenti sull'*operand stack* all'istruzione i del metodo.
- $Z^\sigma[i]$ è una pila di tipi di oggetti di cui il Thread detiene il lock all'istruzione i.
- $T^\sigma[i]$ è l'insieme di tipi di oggetti corrispondenti ai thread mandati in esecuzione dal thread corrente che sono attivi all'istruzione i.
- infine L è l'insieme delle LAMs, quindi il tipo comportamentale, assegnato al nostro programma P^σ

L'applicazione di una mappa parziale G ad un indirizzo i, nelle nostre regole di tipaggio viene abbreviata con G_i.

Il tipo generico τ, oltre al tipo TOP, utlizzato per assegnare un tipo iniziale alla variabili locali di un metodo, comprende anche interi INT, tipi oggetto Θ e tipi di oggetti indicizzati $\widehat{\Theta}$. Questi ultimi sono definiti come:

$$\widehat{\Theta} = \{\sigma_i[a_j : \tau_j^{j\in J}] \mid \sigma[a_j : \tau_j^{j\in J}] \in \Theta \ \text{ and } \ i \ fresh\}$$

Mediante la notazione $\sigma[a_j : \tau_j^{j\in J}]$ intendiamo specificare nome e tipo dei campi di un oggetto di tipo σ.
Per rappresentare i tipi oggetto nelle formule si utilizza σ mentre, per rappresentare i tipi degli oggetti indicizzati, si utilizza σ_i oppure $\widehat{\sigma}$ a seconda che sia necessario/oppurtuno esplicitare o meno l'indice del tipo indicizzato. Il tipo τ^σ, invece, può essere utilizzato per rappresentare un

tipo σ o σ_i. Sia inoltre *Type* una funzione parziale da tipi a tipi definita come segue: $Type[\sigma_i] = \sigma$, e $Type[\tau] = \tau$ altrimenti.

Un indice *fresh* i viene assegnato ad un tipo σ al momento della sua creazione (istruzione *new*). Quando quest'ultimo verrà duplicato mediante un'istruzione *dup* sullo *stack*, l'indice della sua copia rimarrà inalterato. In altre parole, tutte le variabili che hanno lo stesso tipo σ_i sono *aliases*. Per assegnare correttamente l'indice i delle copie, utilizziamo la seguente funzione di indicizzazione τ_i, definita come segue:

1. $\tau_i = \tau$, se $\tau = \text{INT}$ or $\tau = \text{TOP}$ (*solo le copie dei riferimenti sono rilevanti*);
2. $\tau_i = \widehat{\sigma}$, se $\tau = \widehat{\sigma}$ (*successive copie della variabile ne mantengono il tipo della prima copia*).

Per ogni tipo σ definiamo $\bar{\sigma}$ come l'insieme di campi $[a_j : \tau_j^{j \in J}]$ specificati nella definizione di classe corrispondente. Ad esempio per indicare che il campo a della classe di tipo σ ha tipo τ usiamo la notazione $\bar{\sigma}.a : \tau$. Assumiamo che i campi di un oggetto non possano essere modificati (o meglio possano essere modificati assegnando loro lo stesso valore che già possedevano) e pertanto il loro tipo rimarrà sempre lo stesso e l'uso della regola $putfield$ e $putstatic$, per assegnare un valore ad un determinato campo di un oggetto o di una classe, sarà consentita solamente se assegnerà a tale campo lo stesso tipo che questo già possedeva. Se a è un campo di tipo τ_j definito in un oggetto di tipo $\hat{\sigma}$ useremo la notazione $\tau_j^{\hat{\sigma}.a}$ per indicare in modo univoco il tipo immutabile di tale campo. La funzione τ_i se applicata al tipo di un campo si comporterà esattamente come fa quando applicata ai tipi indicizzati cioè restituirà il tipo stesso senza modificarlo.
La regola che prova $F^\sigma, S^\sigma, Z^\sigma, T^\sigma \vdash P^\sigma : L$ è:

$$\frac{\begin{array}{c} F^\sigma[1_\sigma] = F^\sigma_{\text{TOP}} \\ S^\sigma[1_\sigma] = \varepsilon \\ Z^\sigma[1_\sigma] = \varepsilon \\ T^\sigma[1_\sigma] = \varepsilon \\ \forall i \in \mathsf{dom}[P^\sigma].\ F^\sigma, S^\sigma, Z^\sigma, T^\sigma, i \vdash P^\sigma : L_{i_\sigma} \end{array}}{F^\sigma, S^\sigma, Z^\sigma, T^\sigma \vdash P^\sigma : L_{1_\sigma}; \ldots; L_{n_\sigma}}$$

dove:

- l'istruzione 1_σ è la prima istruzione del metodo *run* dell'oggetto thread di tipo σ
- F^σ_{TOP} è la funzione che inizializza l'array delle variabili locali del metodo $run()$ mappando 0 a σ e gli indici delle altre variabili al tipo TOP
- L_i è la LAM dell'istruzione i

Come detto prima ogni flusso di esecuzione di un thread sarà costituito da uno o più invocazioni di metodo. Le regole usate per la tipizzazione di un generico metodo $m(\sigma_{this}[a_j : \tau_j^{j \in J}], \tau_1[a_k : \tau_k^{k \in K}], \ldots, \tau_n[a_w : \tau_w^{w \in W}])$, che provano cioè $F^\sigma_m, S^\sigma_m, Z^\sigma_m, T^\sigma_m \vdash m(\sigma_{this}[a_j : \tau_j^{j \in J}], \tau_1[a_k : \tau_k^{k \in K}], \ldots, \tau_n[a_w : \tau_w^{w \in W}]) : L^m$, sono due a seconda che il metodo sia o meno synchronized. Per un metodo $m(\sigma_{this}[a_j : \tau_j^{j \in J}], \tau_1[a_k : \tau_k^{k \in K}], \ldots, \tau_n[a_w : \tau_w^{w \in W}])$ non synchronized la regola è la seguente:

$$\frac{\begin{array}{c} F^\sigma_m[1_m] = F^m_{\text{TOP}} \\ S^\sigma_m[1_m] = \varepsilon \\ Z^\sigma_m[1_m] = \varepsilon \\ T^\sigma_m[1_m] = \varepsilon \\ \forall i \in \mathsf{dom}[P^m].\ F^\sigma_m, S^\sigma_m, Z^\sigma_m, T^\sigma_m, i \vdash \\ m(\sigma_{this}[a_j : \tau_j^{j \in J}], \tau_1[a_k : \tau_k^{k \in K}], \ldots, \tau_n[a_w : \tau_w^{w \in W}]) : L_{i_m} \end{array}}{\begin{array}{c} F^\sigma_m, S^\sigma_m, Z^\sigma_m, T^\sigma_m \vdash \\ m(\sigma_{this}[a_j : \tau_j^{j \in J}], \tau_1[a_k : \tau_k^{k \in K}], \ldots, \tau_n[a_w : \tau_w^{w \in W}]) : L_{1_m}; \ldots; L_{n_m} \end{array}}$$

mentre per un metodo $m(\sigma_{this}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])$ dichiarato synchronized abbiamo:

$$\frac{\begin{array}{c} F_m^\sigma[1_m] = F_{\text{TOP}}^m \\ S_m^\sigma[1_m] = \varepsilon \\ Z_m^\sigma[1_m] = \sigma_{this} \\ T_m^\sigma[1_m] = \varepsilon \\ \forall i \in \mathsf{dom}[P^m].\ F_m^\sigma, S_m^\sigma, Z_m^\sigma, T_m^\sigma, i \vdash \\ m(\sigma_{this}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) : L_{i_m} \end{array}}{\begin{array}{c} F_m^\sigma, S_m^\sigma, Z_m^\sigma, T_m^\sigma \vdash \\ m(\sigma_{this}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) : L_{1_m}; \ldots; L_{n_m} \end{array}}$$

Come si può vedere, in quest'ultimo caso, a differenza del precedente, la sequenza Z all'inizio del metodo conterrà già il tipo σ_{this} dell'oggetto su cui il metodo è stato invocato (che è il tipo del riferimento *this* che è salvato nella posizione 0 dell'array delle variabili locali). Infine, in entrambi i casi F_{TOP}^m è la funzione che inizializza l'array delle variabili locali mettendo nella posizione 0 il tipo σ_{this}, nelle posizioni seguenti i tipi dei parametri passati al metodo ed infine inizializzando i tipi delle altre variabili locali a TOP.
Infine, consideriamo le funzioni che a partire dalla pila Z e da T calcolano le LAM a seconda delle regole di tipaggio.

$\hat{T} = \{run(\sigma_i)\ \&\ run(\sigma_{i+1}) \mid 1 \leq i \leq n-1\ \wedge\ \hat{\sigma} \in T\}$
$Z = \sigma_n \cdot \ldots \cdot \sigma_1$
$\hat{\bar{Z}} = \{(\sigma_{i+1}, \sigma_i) \mid 1 \leq i \leq n-1\ \wedge\ \sigma_{i+1} \notin [\sigma_{i-1} \cdot \ldots \cdot \sigma_1]\ \wedge\ \sigma_{i+1} \neq \sigma_i\}$
$\bar{Z} = \sigma_n \ldots \sigma_{i+1} \cdot \sigma_i \cdot \sigma_{i-1} \ldots \sigma_1\ \ dove\ \ (\sigma_{i+1},\ \sigma_i),\ (\sigma_i,\ \sigma_{i-1})\ \in\ \hat{\bar{Z}}\quad \forall i\ \ 1 < i < n-1$

dove o_i rappresenta un oggetto e $top(\bar{Z})$ è l'elemento in cima alla pila $\bar{Z}$.

Di seguito vengono mostrate le regole di tipaggio per il giudizio $F^\sigma, S^\sigma, Z^\sigma, T^\sigma, i \vdash P^\sigma : L_i$. In tali regole viene omesso l'apice σ, e si assume che i tipi dei campi di ogni classe del programma P siano noti (essendo presenti nella Constant Pool).

$$\frac{\begin{array}{c} P[i] = \texttt{pop} \\ F_i = F_{i+1} \\ S_i = \tau \cdot S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{iinc} \ \ index \ INT \\ index \in \mathsf{dom}[F_i] \\ F_i[index] = INT \\ F_i = F_{i+1} \\ S_i = S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{iload} \ \ index \\ index \in \mathsf{dom}[F_i] \\ F_i[index] = INT \\ F_i = F_{i+1} \\ INT \cdot S_i = S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{aload}\ \ index \\ index \in \mathsf{dom}[F_i] \\ F_i[index] = \hat{\sigma}[a_j : \tau_j^{j \in J}] \\ F_i = F_{i+1} \\ \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_i = S_{i+1} \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{istore}\ \ index \\ index \in \mathsf{dom}[F_i] \\ F_i[index \mapsto INT] = F_{i+1} \\ S_i = INT \cdot S_{i+1} \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{astore}\ \ index \\ index \in \mathsf{dom}[F_i] \\ F_i[index \mapsto \hat{\sigma}[a_j : \tau_j^{j \in J}]] = F_{i+1} \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_{i+1} \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{goto}\ L \\ F_i = F_L \\ S_i = S_L \\ T_i = T_L \\ L \in \mathsf{dom}[P] \\ Z_i = Z_L \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{aconst\_null} \\ F_i = F_{i+1} \\ Type[\hat{\sigma}] = NULL \\ \hat{\sigma} \cdot S_i = S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \;\&\; \hat{\bar{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{bipush}\ n \\ -128 \leq n \leq 127 \\ F_i = F_{i+1} \\ INT \cdot S_i = S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \;\&\; \hat{\bar{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{iadd} \\ F_i = F_{i+1} \\ S_i = INT \cdot INT \cdot S' \\ S_{i+1} = INT \cdot S' \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \;\&\; \hat{\bar{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{isub} \\ F_i = F_{i+1} \\ S_i = INT \cdot INT \cdot S' \\ S_{i+1} = INT \cdot S' \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \;\&\; \hat{\bar{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{imul} \\ F_i = F_{i+1} \\ S_i = INT \cdot INT \cdot S' \\ S_{i+1} = INT \cdot S' \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{idiv} \\ F_i = F_{i+1} \\ S_i = INT \cdot INT \cdot S' \\ S_{i+1} = INT \cdot S' \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{irem} \\ F_i = F_{i+1} \\ S_i = INT \cdot INT \cdot S' \\ S_{i+1} = INT \cdot S' \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{iconst\_m1} \\ F_i = F_{i+1} \\ INT \cdot S_i = S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{iconst\_}n \\ 0 \leq n \leq 5 \\ F_i = F_{i+1} \\ INT \cdot S_i = S_{i+1} \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{dup} \\ F_i = F_{i+1} \\ S_i = \tau_i[a_j : \tau_j^{j \in J}] \cdot S' \\ S_{i+1} = \tau_i[a_j : \tau_j^{j \in J}] \cdot \tau_i[a_j : \tau_j^{j \in J}] \cdot S' \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{ifne}\ L \\ F_i = F_{i+1} = F_L \\ S_i = INT \cdot S_{i+1} = INT \cdot S_L \\ T_i = T_{i+1} \\ i+1, L \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} = Z_L \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

Le altre istruzioni di salto condizionale (if) viste nella semantica operazionale vengono qui omesse dal momento che le loro premesse e la loro conclusione sono identiche a quelle dell'istruzione $ifne$

$$\frac{\begin{array}{c}P[i] = \texttt{invokespecial} < init > (\sigma_{this}[a_j : \bot^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])VOID \\ F_i = F_{i+1} \\ S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot \hat{\sigma}[a_j : \bot^{j\in J}] \cdot S_{i+1} \\ Type[\hat{\sigma}] = \sigma_{this}[a_j : \tau_j^{j\in J}] \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_{i+1} = Z_i\end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ < init > (\hat{\sigma}[a_j : \bot^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])}$$

$$\frac{\begin{array}{c}P[i] = \texttt{invokevirtual}\ m(\sigma_{this}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])INT \\ F_i = F_{i+1} \\ S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot \hat{\sigma}[a_j : \tau_j^{j\in J}] \cdot S' \\ S_{i+1} = INT \cdot S' \\ Type[\hat{\sigma}[a_j : \tau_j^{j\in J}]] = \sigma_{this}[a_j : \tau_j^{j\in J}] \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_{i+1} = Z_i\end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ m(\hat{\sigma}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) \ \&\ (top(\bar{Z}_i), \hat{\sigma}[a_j : \tau_j^{j\in J}])}$$

$$\frac{\begin{array}{c}P[i] = \texttt{invokevirtual}\ m(\sigma_{this}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])\sigma[a_v : \tau_v^{v\in V}] \\ F_i = F_{i+1} \\ Type[\tau_i[a_v : \tau_v^{v\in V}]] = \sigma[a_v : \tau_v^{v\in V}] \\ S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot \hat{\sigma}[a_j : \tau_j^{j\in J}] \cdot S' \\ S_{i+1} = \tau_x[a_v : \tau_v^{v\in V}] \cdot S' \\ Type[\hat{\sigma}[a_j : \tau_j^{j\in J}]] = \sigma_{this}[a_j : \tau_j^{j\in J}] \\ Type[\tau_x[a_v : \tau_v^{v\in V}]] = \sigma[a_v : \tau_v^{v\in V}] \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_{i+1} = Z_i\end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ m(\hat{\sigma}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) \ \&\ (top(\bar{Z}_i), \hat{\sigma}[a_j : \tau_j^{j\in J}])}$$

$$\frac{\begin{array}{c}P[i] = \texttt{invokevirtual}\ m(\sigma_{this}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])VOID \\ F_i = F_{i+1} \\ S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot \hat{\sigma}[a_j : \tau_j^{j\in J}] \cdot S_{i+1} \\ Type[\hat{\sigma}[a_j : \tau_j^{j\in J}]] = \sigma_{this}[a_j : \tau_j^{j\in J}] \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_{i+1} = Z_i\end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ m(\hat{\sigma}[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) \ \&\ (top(\bar{Z}_i), \hat{\sigma}[a_j : \tau_j^{j\in J}])}$$

$$
\begin{gathered}
P[i] = \texttt{invokestatic}\ \sigma[a_j : \tau_j^{j\in J}].m(\tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])VOID \\
F_i = F_{i+1} \\
S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot S_{i+1} \\
T_i = T_{i+1} \\
i + 1 \in \mathsf{dom}[P] \\
Z_{i+1} = Z_i
\end{gathered}
$$

$$
F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ m(\sigma[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) \ \&\ (top(\bar{Z}_i), \sigma[a_j : \tau_j^{j\in J}])
$$

$$
\begin{gathered}
P[i] = \texttt{invokestatic}\ \sigma[a_j : \tau_j^{j\in J}].m(\tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])INT \\
F_i = F_{i+1} \\
S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot S' \\
S_{i+1} = INT \cdot S' \\
T_i = T_{i+1} \\
i + 1 \in \mathsf{dom}[P] \\
Z_{i+1} = Z_i
\end{gathered}
$$

$$
F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ m(\sigma[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) \ \&\ (top(\bar{Z}_i), \sigma[a_j : \tau_j^{j\in J}])
$$

$$
\begin{gathered}
P[i] = \texttt{invokestatic}\ \sigma[a_j : \tau_j^{j\in J}].m(\tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}])\sigma'[a_v : \tau_v^{v\in V}] \\
F_i = F_{i+1} \\
S_i = \tau_n[a_w : \tau_w^{w\in W}] \cdot \ldots \cdot \tau_1[a_k : \tau_k^{k\in K}] \cdot S' \\
S_{i+1} = \tau_x[a_v : \tau_v^{v\in V}] \cdot S' \\
Type[\tau_x[a_v : \tau_v^{v\in V}]] = \sigma'[a_v : \tau_v^{v\in V}] \\
T_i = T_{i+1} \\
i + 1 \in \mathsf{dom}[P] \\
Z_{i+1} = Z_i
\end{gathered}
$$

$$
F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ m(\sigma[a_j : \tau_j^{j\in J}], \tau_1[a_k : \tau_k^{k\in K}], \ldots, \tau_n[a_w : \tau_w^{w\in W}]) \ \&\ (top(\bar{Z}_i), \sigma[a_j : \tau_j^{j\in J}])
$$

$$\frac{\begin{array}{c}P[i] = \texttt{invokevirtual}\ \ java/lang/Thread.start() \\ F_i = F_{i+1} \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \\ T_{i+1} = T_i \uplus \{\sigma[a_j : \hat{\tau}_j^{j \in J}]\}\end{array}}{F, S, Z, T, i \vdash P : \hat{\bar{Z}}_i \ \&\ \hat{T}_i}$$

$$\frac{\begin{array}{c}P[i] = \texttt{invokevirtual}\ \ java/lang/Thread.join() \\ F_i = F_{i+1} \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \\ T_{i+1} = T_i \setminus \{\hat{\sigma}\}\end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i \ \&\ (top(\bar{Z}), \hat{\sigma}[a_j : \tau_j^{j \in J}]}$$

$$\frac{\begin{array}{c}P[i] = \texttt{new}\ \ \sigma[a_j : \bot^{j \in J}] \\ F_i = F_{i+1} \\ S_{i+1} = \sigma_i[a_j : \bot^{j \in J}] \cdot S_i \\ T_i = T_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1}\end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\bar{Z}}_i}$$

$$\frac{P[i] = \texttt{return}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{P[i] = \texttt{ireturn} \quad S_i = INT \cdot S'}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{P[i] = \texttt{areturn} \quad S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S'}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{ldc}\ INT \\ F_i = F_{i+1} \\ S_{i+1} = INT \cdot S_i \\ i + 1 \in \mathsf{dom}[P] \\ T_i = T_{i+1} \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{ldc}\ \sigma[a_j : \tau_j^{j \in J}] \\ F_i = F_{i+1} \\ S_{i+1} = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_i \\ Type[\hat{\sigma}[a_j : \tau_j^{j \in J}]] = \sigma[a_j : \tau_j^{j \in J}] \\ i + 1 \in \mathsf{dom}[P] \\ T_i = T_{i+1} \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

Come si può vedere dalle regola *putfield* e *putstatic* si assume che i campi di un oggetto possono essere modificati solo assegnando loro lo stesso valore o se non inizializzati. Il loro tipo pertanto non potrà variare.

$$\frac{\begin{array}{c} P[i] = \texttt{putfield}\ \sigma.a : \tau \\ \bar{\sigma}.a : \tau_k \\ F_i = F_{i+1} \\ S_i = \tau_x \cdot \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_{i+1} \\ Type[\hat{\sigma}[a_j : \tau_j^{j \in J}]] = \sigma[a_j : \tau_j^{j \in J}] \\ \tau_k = \bot \vee \tau_k = \tau_x \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{getfield}\ \sigma.a : \tau \\ \bar{\sigma}.a : \tau_x \\ F_i = F_{i+1} \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S' \\ S_{i+1} = \tau_x \cdot S' \\ Type[\hat{\sigma}[a_j : \tau_j^{j \in J}]] = \sigma[a_j : \tau_j^{j \in J}] \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{putstatic}\ \sigma.a : \tau \\ \bar{\sigma}.a : \tau_k \\ F_i = F_{i+1} \\ S_i = \tau_x \cdot S_{i+1} \\ \tau_k = \bot \vee \tau_k = \tau_x \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{getstatic}\ \sigma.a : \tau \\ \bar{\sigma}.a : \tau_x \\ F_i = F_{i+1} \\ S_{i+1} = \tau_x \cdot S_i \\ T_i = T_{i+1} \\ i + 1 \in \mathsf{dom}[P] \\ Z_i = Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \&\ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{monitorenter} \\ F_i = F_{i+1} \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ T_i = T_{i+1} \\ Z_{i+1} = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot Z_i \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{monitorexit} \\ F_i = F_{i+1} \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S_{i+1} \\ i+1 \in \mathsf{dom}[P] \\ T_i = T_{i+1} \\ Z_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot Z_{i+1} \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

Infine vediamo le regole legate al lancio delle eccezioni. Qui la notazione $P_{ExcTable}(i, \sigma)$ restituisce l'indirizzo dell'$ExceptionHandler$ delegato alla gestione dell'eccezione di tipo σ quando questa si verifica all'istruzione i del metodo oppure $\perp$ se l'eccezione non è gestita.

$$\frac{\begin{array}{c} P[i] = \texttt{athrow} \\ P_{ExcTable}(i, \hat{\sigma}[a_j : \tau_j^{j \in J}]) = k \\ F_i = F_k \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S' \\ S_k = \hat{\sigma}[a_j : \tau_j^{j \in J}] \\ T_i = T_k \\ k \in \mathsf{dom}[P] \\ Z_i = Z_k \\ Type[\hat{\sigma}[a_j : \tau_j^{j \in J}]] = java/lang/Throwable \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

$$\frac{\begin{array}{c} P[i] = \texttt{athrow} \\ P_{ExcTable}(i, \hat{\sigma}[a_j : \tau_j^{j \in J}]) = \perp \\ S_i = \hat{\sigma}[a_j : \tau_j^{j \in J}] \cdot S' \\ Type[\hat{\sigma}[a_j : \tau_j^{j \in J}]] = java/lang/Throwable \end{array}}{F, S, Z, T, i \vdash P : \hat{T}_i \ \& \ \hat{\hat{Z}}_i}$$

Di seguito mostriamo come, partendo da un programma Java e analizzandone le relative istruzioni bytecode, è possibile assegnare ad ogni metodo la relativa LAM ed andare poi a cercare eventuali circolarità che segnalino la presenza di un possibile deadlock.

```
package deadlock;

// pc1 = monitorenter , pc2 = monitorenter

public class Deadlock1 {

   private Object a = new Object();
   private Object b = new Object();


   public void takelock() throws InterruptedException{
      synchronized (a) {
         synchronized (b) {
         }
      }
   }

   public void takelock2() throws InterruptedException{
      synchronized (b) {
         synchronized (a) {
         }
      }
   }

   public static void main(String[]args) throws InterruptedException{

      Deadlock1 d = new Deadlock1();


      Thread t1 = new Thread(){

         @Override
         public void run() {
            try {
               d.takelock();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      };

      t1.start();

      Thread t2 = new Thread() {

         @Override
         public void run() {
            try {
```

```
            d.takelock2();
          } catch (InterruptedException e) {
            e.printStackTrace();
          }
        }
    };

    t2.start();
  }
}
```

Vediamo di seguito, per ogni metodo, il relativo bytecode e la LAM corrispondente. Nei parametri dei metodi viene usata la notazione $\tau_{[i]}$ per indicare il tipo τ del parametro formale in posizione i nella lista dei parametri. Tale tipo verrà trattato come se fosse un tipo già indicizzato τ_i e quindi non verrà modificato da un eventuale load (o dup) del tipo sullo stack.
A partire dalla LAM del metodo main si andrà ad identificare una circolarità che indica il possibile stato di deadlock del programma. Per semplicità nel bytecode viene omesso il codice relativo alla gestione delle eccezioni ed utilizzata le seguenti abbreviazioni:

- D per $Deadlock1$.
- O per $Object$ (che nel bytecode è tradotto in $lava/lang/Object$).
- $T1$ per $Deadlock1\$1$ cioè il primo Thread.
- $T2$ per $Deadlock1\$2$ cioè il secondo Thread.

i	$P[i]$	S_i	T_i
1	`new` D	ϵ	$\emptyset$
2	`dup`	$D_2[a:\perp, b:\perp]$	$\emptyset$
3	`invokespecial` $< init >(D)$	$D_2[a:\perp, b:\perp] \cdot D_2[a:\perp, b:\perp]$	$\emptyset$
4	**astore_1**	$D_2[a:O_3, b:O_4]$	$\emptyset$
5	`new` $T1$	ϵ	$\emptyset$
6	`dup`	$T1_5[d:\perp]$	$\emptyset$
7	**aload_1**	$T1_5[d:\perp] \cdot T1_5[d:\perp]$	$\emptyset$
8	`invokespecial` $< init >(T1, D)$	$D_2[a:O_3, b:O_4] \cdot T1_5[d:\perp] \cdot T1_5[d:\perp]$	$\emptyset$
9	**astore_2**	$T1_5[d:D_2]$	$\emptyset$
10	**aload_2**	ϵ	$\emptyset$
11	`invokevirtual` $start()$	$T1_5[d:D_2]$	$\emptyset$
12	`new` $T2$	ϵ	$\{T1_5[d:D_2]\}$
13	`dup`	$T2[d:\perp]$	$\{T1_5[d:D_2]\}$
14	**aload_1**	$T2[d:\perp] \cdot T2[d:\perp]$	$\{T1_5[d:D_2]\}$
15	`invokespecial` $< init >(T2, D)$	$D_2[a:O_3, b:O_4] \cdot T2[d:\perp] \cdot T2[d:\perp]$	$\{T1_5[d:D_2]\}$
16	**astore_3**	$T2_6[d:D_2]$	$\{T1_5[d:D_2]\}$
17	**aload_3**	ϵ	$\{T1_5[d:D_2]\}$
18	`invokevirtual` $start()V$	$T2_6[d:D_2]$	$\{T1_5[d:D_2]\}$
19	`return`	ϵ	$\{T1_5[d:D_2], T2[d:D_2]\}$

i	$F[0]$	$F_i[1]$	$F_i[2]$	$F_i[3]$
1	TOP	TOP	TOP	TOP
2	TOP	TOP	TOP	TOP
3	TOP	TOP	TOP	TOP
4	TOP	TOP	TOP	TOP
5	TOP	$D_2[a:O_3, b:O_4]$	TOP	TOP
6	TOP	$D_2[a:O_3, b:O_4]$	TOP	TOP
7	TOP	$D_2[a:O_3, b:O_4]$	TOP	TOP
8	TOP	$D_2[a:O_3, b:O_4]$	TOP	TOP
9	TOP	$D_2[a:O_3, b:O_4]$	TOP	TOP
10	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
11	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
12	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
13	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
14	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
15	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
16	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	TOP
17	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	$T2_6[d:D_2]$
18	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	$T2_6[d:D_2]$
19	TOP	$D_2[a:O_3, b:O_4]$	$T1_5[d:D_2]$	$T2_6[d:D_2]$

$$\text{LAM(main(String[]))} =$$

$$\hat{T}_1 \;\&\; \hat{\hat{Z}}_1 \;;\; \hat{T}_2 \;\&\; \hat{\hat{Z}}_2 \;;\; \hat{T}_3 \;\&\; \hat{\hat{Z}}_3 \;\&\; < init > (D_2[a:\perp, b:\perp]) \;;\; \hat{T}_4 \;\&\; \hat{\hat{Z}}_4 \;;\; \hat{T}_5 \;\&\; \hat{\hat{Z}}_5 \;;\; \hat{T}_6 \;\&\; \hat{\hat{Z}}_6 \;;\; \hat{T}_7 \;\&\; \hat{\hat{Z}}_7 \;;\; \hat{T}_8 \;\&\; \hat{\hat{Z}}_8 \;\&\; < init > (T1_5[d:\perp], D_2[a:O_3, b:O_4]) \;;\; \hat{T}_9 \;\&\; \hat{\hat{Z}}_9 \;;\; \hat{T}_{10} \;\&\; \hat{\hat{Z}}_{10} \;;\; \hat{T}_{11} \;\&\; \hat{\hat{Z}}_{11} \;;\; \hat{T}_{12} \;\&\; \hat{\hat{Z}}_{12} \;;\; \hat{T}_{13} \;\&\; \hat{\hat{Z}}_{13} \;;\; \hat{T}_{14} \;\&\; \hat{\hat{Z}}_{14} \;;\; \hat{T}_{15} \;\&\; \hat{\hat{Z}}_{15} \;\&\; < init > (T2_6[d:\perp], D_2[a:O_3, b:O_4]) \;;\; \hat{T}_{16} \;\&\; \hat{\hat{Z}}_{16} \;;\; \hat{T}_{17} \;\&\; \hat{\hat{Z}}_{17} \;;\; \hat{T}_{18} \;\&\; \hat{\hat{Z}}_{18} \;;\; \hat{T}_{19} \;\&\; \hat{\hat{Z}}_{19} \;;$$

$$= 0 \;;0 \;;< init > (D_2[a:\perp, b:\perp]) \;;\; 0 \;;\; 0 \;;\; 0 \;;\; 0 \;;\; < init > (T1_5[d:\perp], D_2[a:O_3, b:O_4]) \;;\; 0 \;;\; 0 \;;\; 0 \;;$$
$$run(T1_5[d:D_2]) \;;\; run(T1_5[d:D_2]) \;;\; run(T1_5[d:D_2]) \;;$$
$$run(T1_5[d:D_2]) \;\&\; < init > (T2_6[d:\perp], D_2[a:O_3, b:O_4]) \;;$$
$$run(T1_5[d:D_2]) \;;\; run(T1_5[d:D_2]) \;;\; run(T1_5[d:D_2]) \;;\; run(T1_5[d:D_2]) \;\&\; run(T2_6[d:D_2]) \;\;;$$

$$=< init > (D_2[a:\perp, b:\perp]) \;;\; < init > (T1_5[d:\perp], D_2[a:O_3, b:O_4]) \;;\; run(T1_5[d:D_2]) \;;$$
$$run(T1_5[d:D_2]) \;\&\; < init > (T2_6[d:\perp], D_2[a:O_3, b:O_4]) \;;\; run(T1_5[d:D_2]) \;\&\; run(T2_6[d:D_2]) \;\;;$$

Fig. 1. Metodo main() che lancia i due thread e relativa LAM

i	$P[i]$	$F_i[0]$	S_i
1	**aload_0**	$D_{[0]}[a:\perp, b:\perp]$	ϵ
2	`invokespecial` $< init > (O)V$	$D_{[0]}[a:\perp, b:\perp]$	$D_{[0]}[a:\perp, b:\perp]$
3	**aload_0**	$D_{[0]}[a:\perp, b:\perp]$	ϵ
4	`new` O	$D_{[0]}[a:\perp, b:\perp]$	$D_{[0]}[a:\perp, b:\perp]$
5	`dup`	$D_{[0]}[a:\perp, b:\perp]$	$O_7^{\perp} \cdot D_{[0]}[a:\perp, b:\perp]$
6	`invokespecial` $< init > (O)V$	$D_{[0]}[a:\perp, b:\perp]$	$O_7^{\perp} \cdot O_7^{\perp} \cdot D_{[0]}[a:\perp, b:\perp]$
7	`putfield` $D.a : O$	$D_{[0]}[a:\perp, b:\perp]$	$O_3 \cdot D_{[0]}[a:\perp, b:\perp]$
8	**aload_0**	$D_{[0]}[a:O_3, b:\perp]$	ϵ
9	`new` O	$D_{[0]}[a:O_3, b:\perp]$	$D_{[0]}[a:O_3, b:\perp]$
10	`dup`	$D_{[0]}[a:O_3, b:\perp]$	$O_8^{\perp} \cdot D_{[0]}[a:O_3, b:\perp]$
11	`invokespecial` $< init > (O)V$	$D_{[0]}[a:O_3, b:\perp]$	$O_8^{\perp} \cdot O_8^{\perp} \cdot D_{[0]}[a:O_3, b:\perp]$
12	`putfield` $D.b : O$	$D_{[0]}[a:O_3, b:\perp]$	$O_4 \cdot D_{[0]}[a:O_3, b:\perp]$
13	`return`	$D_{[0]}[a:O_3, b:O_4]$	ϵ

$$\text{LAM}(< init > (D_{[0]}[a:\perp, b:\perp])) =$$
$$\hat{T}_1 \;\&\; \hat{\tilde{Z}}_1 \;;\; \hat{T}_2 \;\&\; \hat{\tilde{Z}}_2 \;;\; \hat{T}_3 \;\&\; \hat{\tilde{Z}}_3 \;;\; \hat{T}_4 \;\&\; \hat{\tilde{Z}}_4 \;;\; \hat{T}_5 \;\&\; \hat{\tilde{Z}}_5 \;;\; \hat{T}_6 \;\&\; \hat{\tilde{Z}}_6 \;\&\; < init > (O_7^{\perp});\; \hat{T}_7 \;\&\; \hat{\tilde{Z}}_7 \;;\; \hat{T}_8 \;\&\; \hat{\tilde{Z}}_8 \;;\; \hat{T}_9 \;\&\; \hat{\tilde{Z}}_9 \;;\; \hat{T}_{10} \;\&\; \hat{\tilde{Z}}_{10} \;;$$
$$\hat{T}_{11} \;\&\; \hat{\tilde{Z}}_{11} \;\&\; < init > (O_8^{\perp});\; \hat{T}_{12} \;\&\; \hat{\tilde{Z}}_{12} \;;\; \hat{T}_{13} \;\&\; \hat{\tilde{Z}}_{13} \;;$$

$$= 0;\; 0;\; 0;\; 0;\; 0;\; < init > (O_7^{\perp});\; 0;\; 0;\; 0;\; 0;\; < init > (O_8^{\perp});\; 0;\; 0;$$

$$= < init > (O_7^{\perp});\;\; < init > (O_8^{\perp});\; = 0$$

Fig. 2. Metodo $< init > (D_{[0]}[a:\perp, b:\perp])$ e relativa LAM

A riga 2 del precedente codice viene invocato il costruttore della superclasse $java/lang/Object$; Nelle LAM non prendiamo in considerazione la relativa invocazione $< init > (java/lang/Object)V$ perchè chiama il costruttore di una classe definita nella libreria standard di Java e non in una delle classi del nostro programma. Lo stesso facciamo con l'invocazione $< init > (java/lang/Thread)V$ dei due metodi seguenti.

i	$P[i]$	$F_i[0]$	$F_i[1]$	S_i
1	**aload_0**	$T1_{[0]}[d:\perp]$	$D_{[1]}[a:O_9, b:O_{10}]$	ϵ
2	**aload_1**	$T1_{[0]}[d:\perp]$	$D_{[1]}[a:O_9, b:O_{10}]$	$T1_{[0]}[d:\perp]$
3	`putfield` $T1.d:D$	$T1_{[0]}[d:\perp]$	$D_{[1]}[a:O_9, b:O_{10}]$	$D_{[1]}[a:O_9, b:O_{10}] \cdot T1_{[0]}[d:\perp]$
4	**aload_0**	$T1_{[0]}[d:D_{[1]}]$	$D_{[1]}[a:O_9, b:O_{10}]$	ϵ
5	`invokespecial`$< init >(java/lang/Thread)V$	$T1_{[0]}[d:D_{[1]}]$	$D_{[1]}[a:O_9, b:O_{10}]$	$T1_{[0]}$
6	`return`	$T1_{[0]}[d:D_{[1]}]$	$D_{[1]}[a:O_9, b:O_{10}]$	ϵ

$$\text{LAM}(< init > (T1_{[0]}[d:\perp], D_{[1]}[a:O_9, b:O_{10}]))= \ \hat{T}_1 \ \& \ \hat{\hat{Z}}_1 \ ; \ \hat{T}_2 \ \& \ \hat{\hat{Z}}_2 \ ; \ \hat{T}_3 \ \& \ \hat{\hat{Z}}_3 \ ; \ \hat{T}_4 \ \& \ \hat{\hat{Z}}_4 \ ; \ \hat{T}_5 \ \& \ \hat{\hat{Z}}_5 \ ; \ \hat{T}_6 \ \& \ \hat{\hat{Z}}_6 \ ;$$

$$= 0; 0; 0; 0; 0; 0; \ = \ 0;$$

Fig. 3. Metodo $< init > (T1_{[0]}[d:\perp], D_{[1]}[a:O_9, b:O_{10}])$ del primo thread e relativa LAM

i	$P[i]$	$F_i[0]$	$F_i[1]$	S_i
1	**aload_0**	$T2_{[0]}[d:\perp]$	$D_{[1]}[a:O_{11}, b:O_{12}]$	ϵ
2	**aload_1**	$T2_{[0]}[d:\perp]$	$D_{[1]}[a:O_{11}, b:O_{12}]$	$T2_{[0]}[d:\perp]$
3	`putfield` $T2.d:D$	$T2_{[0]}[d:\perp]$	$D_{[1]}[a:O_{11}, b:O_{12}]$	$D_{[1]}[a:O_{11}, b:O_{12}] \cdot T2_{[0]}[d:\perp]$
4	**aload_0**	$T2_{[0]}[d:D_{[1]}]$	$D_{[1]}[a:O_{11}, b:O_{12}]$	ϵ
5	`invokespecial`$< init >(java/lang/Thread)V$	$T2_{[0]}[d:D_{[1]}]$	$D_{[1]}[a:O_{11}, b:O_{12}]$	$T2_{[0]}$
6	`return`	$T2_{[0]}[d:D_{[1]}]$	$D_{[1]}[a:O_{11}, b:O_{12}]$	ϵ

$$\text{LAM}(< init > (T2_{[0]}[d:\perp], D_{[1]}[a:O_{11}, b:O_{12}]))= \ \hat{T}_1 \ \& \ \hat{\hat{Z}}_1 \ ; \ \hat{T}_2 \ \& \ \hat{\hat{Z}}_2 \ ; \ \hat{T}_3 \ \& \ \hat{\hat{Z}}_3 \ ; \ \hat{T}_4 \ \& \ \hat{\hat{Z}}_4 \ ; \ \hat{T}_5 \ \& \ \hat{\hat{Z}}_5 \ ; \ \hat{T}_6 \ \& \ \hat{\hat{Z}}_6 \ ;$$

$$= 0; 0; 0; 0; 0; 0; \ = \ 0;$$

Fig. 4. Metodo $< init > (T2_{[0]}[d:\perp], D_{[1]}[a:O_{11}, b:O_{12}])$ del secondo thread e relativa LAM

i	$P[i]$	$F_i[0]$	S_i	Z_i
1	**aload_0**	$T1_{[0]}[d : D_{13}]$	ϵ	ϵ
2	`getfield` $T1.d : D$	$T1_{[0]}[d : D_{13}]$	$T1_{[0]}[d : D_{13}]$	ϵ
3	`invokevirtual` $takelock(D)V$	$T1_{[0]}[d : D_{13}]$	$D_{13}^{T1_{[0]}.d}$	ϵ
4	`return`	$T1_{[0]}[d : D_{13}]$	ϵ	ϵ

$$\mathrm{LAM}(\mathrm{run}(T1_{[0]}[d : D_{13}])) =$$
$$\hat{T}_1 \ \& \ \hat{\hat{Z}}_1 \ ; \ \hat{T}_2 \ \& \ \hat{\hat{Z}}_2 \ ; \ \hat{T}_3 \ \& \ \hat{\hat{Z}}_3 \ \& \ takelock(D_{13}^{T1_{[0]}.d}) \ \& \ (top(\bar{Z}), D_{13}^{T1_{[0]}.d}); \ \hat{T}_4 \ \& \ \hat{\hat{Z}}_4 \ ;$$
$$= 0; 0; takelock(D_{13}^{T1_{[0]}.d}) \ ; 0;$$
$$= takelock(D_{13}^{T1_{[0]}.d}) \ ;$$

Fig. 5. Metodo $run(T1_{[0]}[d : D_{13}])$ del primo thread e relativa LAM

i	$P[i]$	$F_i[0]$	S_i	Z_i
1	**aload_0**	$T2_{[0]}[d : D_{14}]$	ϵ	ϵ
2	`getfield` $T2.d : D$	$T2_{[0]}[d : D_{14}]$	$T2_{[0]}[d : D_{14}]$	ϵ
3	`invokevirtual` $takelock(D)V$	$T2_{[0]}[d : D_{14}]$	$D_{14}^{T2_{[0]}.d}$	ϵ
4	`return`	$T2_{[0]}[d : D_{14}]$	ϵ	ϵ

$$\mathrm{LAM}(\mathrm{run}(T2_{[0]}[d : D_{14}])) =$$
$$\hat{T}_1 \ \& \ \hat{\hat{Z}}_1 \ ; \ \hat{T}_2 \ \& \ \hat{\hat{Z}}_2 \ ; \ \hat{T}_3 \ \& \ \hat{\hat{Z}}_3 \ \& \ takelock(D_{14}^{T2_{[0]}.d}) \ \& \ (top(\bar{Z}), D_{14}^{T2_{[0]}.d}); \ \hat{T}_4 \ \& \ \hat{\hat{Z}}_4 \ ;$$
$$= 0; 0; takelock(D_{14}^{T2_{[0]}.d}) \ ; 0;$$
$$= takelock(D_{14}^{T2_{[0]}.d}) \ ;$$

Fig. 6. Metodo $run(T2_{[0]}[d : D_{14}])$ del secondo thread e relativa LAM

Nelle regole di seguito σ' e σ'' stanno rispettivamente per $O_{15}^{D_{[0]}.a}$ e $O_{16}^{D_{[0]}.b}$:

i	$P[i]$	$F_i[0]$	$F_i[1]$	$F_i[2]$	S_i	Z_i
1	**aload_0**	$D_{[0]}[a:O_{15}, b:O_{16}]$	TOP	TOP	ϵ	ϵ
2	`getfield` $D.a:O$	$D_{[0]}[a:O_{15}, b:O_{16}]$	TOP	TOP	$D_{[0]}[a:O_{15}, b:O_{16}]$	ϵ
3	`dup`	$D_{[0]}[a:O_{15}, b:O_{16}]$	TOP	TOP	σ'	ϵ
4	**astore_1**	$D_{[0]}[a:O_{15}, b:O_{16}]$	TOP	TOP	$\sigma' \cdot \sigma'$	ϵ
5	`monitorenter`	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	TOP	σ'	ϵ
6	**aload_0**	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	TOP	ϵ	σ'
7	`getfield` $D.b:O$	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	TOP	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'
8	`dup`	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	TOP	σ''	σ'
9	**astore_2**	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	TOP	$\sigma'' \cdot \sigma''$	σ'
10	`monitorenter`	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	σ''	σ''	σ'
11	**aload_2**	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	σ''	ϵ	$\sigma'' \cdot \sigma'$
12	`monitorexit`	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	σ''	σ''	$\sigma'' \cdot \sigma'$
13	**aload_1**	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	σ''	ϵ	σ'
14	`monitorexit`	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	σ''	σ'	σ'
15	`return`	$D_{[0]}[a:O_{15}, b:O_{16}]$	σ'	σ''	ϵ	ϵ

$$\text{LAM}(\text{takelock}(D_{[0]}[a:O_{15}, b:O_{16}]))=$$
$$\hat{T}_1 \,\&\, \hat{\hat{Z}}_1 \;;\; \hat{T}_2 \,\&\, \hat{\hat{Z}}_2 \;;\; \hat{T}_3 \,\&\, \hat{\hat{Z}}_3 \;;\; \hat{T}_4 \,\&\, \hat{\hat{Z}}_4 \;;\; \hat{T}_5 \,\&\, \hat{\hat{Z}}_5 \;;\; \hat{T}_6 \,\&\, \hat{\hat{Z}}_6 \;;\; \hat{T}_7 \,\&\, \hat{\hat{Z}}_7 \;;\; \hat{T}_8 \,\&\, \hat{\hat{Z}}_8 \;;$$
$$\hat{T}_9 \,\&\, \hat{\hat{Z}}_9 \;;\; \hat{T}_{10} \,\&\, \hat{\hat{Z}}_{10};\; \hat{T}_{11} \,\&\, \hat{\hat{Z}}_{11} \;;\; \hat{T}_{12} \,\&\, \hat{\hat{Z}}_{12} \;;\; \hat{T}_{13} \,\&\, \hat{\hat{Z}}_{13} \;;\; \hat{T}_{14} \,\&\, \hat{\hat{Z}}_{14} \;;\; \hat{T}_{15} \,\&\, \hat{\hat{Z}}_{15} \;;$$

$$= 0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;\; (\sigma', \sigma'') \;;\; (\sigma', \sigma'') \;;\; 0 \;;0 \;;0 \;;$$

$$= (\sigma', \sigma'');$$

Fig. 7. Metodo $takelock(D_{[0]}[a:O_{15}, b:O_{16}])$ eseguito dal primo thread

i	$P[i]$	$F_i[0]$	$F_i[1]$	$F_i[2]$	S_i	Z_i
1	**aload_0**	$D_{[0]}[a:O_{17}, b:O_{18}]$	TOP	TOP	ϵ	ϵ
2	`getfield` $D.b:O$	$D_{[0]}[a:O_{17}, b:O_{18}]$	TOP	TOP	$D_{[0]}[a:O_{17}, b:O_{18}]$	ϵ
3	`dup`	$D_{[0]}[a:O_{17}, b:O_{18}]$	TOP	TOP	σ''	ϵ
4	**astore_1**	$D_{[0]}[a:O_{17}, b:O_{18}]$	TOP	TOP	$\sigma'' \cdot \sigma''$	ϵ
5	`monitorenter`	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	TOP	σ''	ϵ
6	**aload_0**	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	TOP	ϵ	σ''
7	`getfield` $D.a:O$	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	TOP	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''
8	`dup`	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	TOP	σ'	σ''
9	**astore_2**	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	TOP	$\sigma' \cdot \sigma'$	σ''
10	`monitorenter`	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	σ'	σ'	σ''
11	**aload_2**	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	σ'	ϵ	$\sigma' \cdot \sigma''$
12	`monitorexit`	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	σ'	σ'	$\sigma' \cdot \sigma''$
13	**aload_1**	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	σ'	ϵ	σ''
14	`monitorexit`	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	σ'	σ''	σ''
15	`return`	$D_{[0]}[a:O_{17}, b:O_{18}]$	σ''	σ'	ϵ	ϵ

$$\mathrm{LAM}(\mathrm{takelock2}(D_{[0]}[a:O_{17}, b:O_{18}]))=$$
$$\hat{T}_1 \;\&\; \hat{\hat{Z}}_1 \;;\; \hat{T}_2 \;\&\; \hat{\hat{Z}}_2 \;;\; \hat{T}_3 \;\&\; \hat{\hat{Z}}_3 \;;\; \hat{T}_4 \;\&\; \hat{\hat{Z}}_4 \;;\; \hat{T}_5 \;\&\; \hat{\hat{Z}}_5 \;;\; \hat{T}_6 \;\&\; \hat{\hat{Z}}_6 \;;\; \hat{T}_7 \;\&\; \hat{\hat{Z}}_7 \;;\; \hat{T}_8 \;\&\; \hat{\hat{Z}}_8 \;;$$
$$\hat{T}_9 \;\&\; \hat{\hat{Z}}_9 \;;\; \hat{T}_{10} \;\&\; \hat{\hat{Z}}_{10};\; \hat{T}_{11} \;\&\; \hat{\hat{Z}}_{11} \;;\; \hat{T}_{12} \;\&\; \hat{\hat{Z}}_{12} \;;\; \hat{T}_{13} \;\&\; \hat{\hat{Z}}_{13} \;;\; \hat{T}_{14} \;\&\; \hat{\hat{Z}}_{14} \;;\; \hat{T}_{15} \;\&\; \hat{\hat{Z}}_{15} \;;$$

$$= 0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;0 \;;\; (\sigma'', \sigma') \;;\; (\sigma'', \sigma') \;;\; 0 \;;0 \;;0 \;;$$

$$= (\sigma'', \sigma');$$

Fig. 8. Metodo *takelock*2($D_{[0]}[a:O_{17}, b:O_{18}]$) eseguito dal secondo thread

Adesso che abbiamo sia la LAM del main che quelle dei metodi del programma andiamo a sostituire nella LAM del main le invocazioni di metodo con le corrispondenti LAM facendo attenzione a rimpiazzare opportunamente i parametri formali con quelli attuali.

$$< init > (D_2[a : \bot, b : \bot]) \ ; \ < init > (T1_5[d : \bot], D_2[a : O_3, b : O_4]) \ ; \ run(T1_5[d : D_2[a : O_3, b : O_4]]) \ ; run(T1_5[d : D_2[a : O_3, b : O_4]]) \ \& \ < init > (T2_6[d : \bot], D_2[a : O_3, b : O_4]) \ ; \ run(T1_5[d : D_2[a : O_3, b : O_4]]) \ \& \ run(T2_6[d : D_2[a : O_3, b : O_4]]) \ ;$$

$$= 0; \ 0; \ takelock(D_2^{T1_5.d}[a : O_3, b : O_4]); takelock(D_2^{T1_5.d}[a : O_3, b : O_4]) \ \& \ 0; \ takelock(D_2^{T1_5.d}[a : O_3, b : O_4]) \ \& \ takelock2(D_2^{T2_6.d}[a : O_3, b : O_4]) \ ;$$

$$= (O_3^{(D_2.a)}, O_4^{(D_2.b)}) \ ; \ (O_3^{(D_2.a)}, O_4^{(D_2.b)}) \ \& \ (O_4^{(D_2.b)}, O_3^{(D_2.a)}) \ ;$$

$$= (O_3^{(D_2.a)}, O_4^{(D_2.b)}) \ \& \ (O_4^{(D_2.b)}, O_3^{(D_2.a)}) \ ;$$

Quindi si ha una circolarità:

$$(O_3^{(D_2.a)}, O_3^{(D_2.a)}) \ ;$$

che indica la presenza di deadlock.

6 Conclusioni

A seguito di una lunga fase progettuale circa la semantica operazionale ed i tipi comportamentali (seppur momentaneamente non considerando alcune primitive quali la *wait* e *notify*) si è potuti giungere all'implementazione di un primo prototipo funzionanate per la costruzione delle LAMs a partire da un semplice programma in Java. Il progetto, dunque, seppur soggetto ad alcune limitazioni, costituisce un primo tentativo per la stesura di un framework generale per l'analisi di deadlock in Java.

References

[1] Elena Giachino, Carlo A Grazia, Cosimo Laneve, Michael Lienhardt, and Peter YH Wong. Deadlock analysis of concurrent objects: Theory and practice. In *Integrated Formal Methods*, pages 394–411. Springer, 2013.

[2] Elena Giachino, Cosimo Laneve, and Michael Lienhardt. A framework for deadlock detection in abs. *Software and System Modeling (to appear, 2014)*, 2014.