UNIVERSITÀ DI PISA

Master of Science in Computer Engineering

Project Report

Secure Information Flow by Model Checking

Formal Methods for Secure Systems

T. Billi, E. Casapieri, A. Di Donato

Academic Year 2019/2020

Indice

Capitolo 1

Introduzione

Il Model Checking è un metodo automatizzato per l'analisi della proprietà dei sistemi. Si tratta di una tecnica di verifica basata, nel caso dei programmi, su: un control flow graph che rappresenta la struttura del programma ed un transition system che rappresenta l'informazione sulla struttura di controllo. La verifica viene effettuata esprimendo una proprietà desiderata del sistema come formula logica e verificando poi con algoritmi di model checking che il modello soddisfi tale formula; viene quindi fornito un metodo di verifica automatica delle proprietà del sistema. Questo approccio ha l'inconveniente che sistemi composti da più sottosistemi possono essere associati ad un modello di stato finito con un numero di stati esponenziale nel numero di sottoinsiemi. Inoltre, i sistemi che sono fortemente dipendenti dai valori dei dati, condividono lo stesso problema producendo un numero di stati esponenziale nel numero di variabili dei dati. Tale problema detto *dell'Esplosione Spaziale degli stati* è stato la causa per il quale gli esempi riportati più avanti sono composti da un numero di istruzioni limitato; simulando esempi con un numero di istruzioni maggiore si è infatti riscontrato un significativo aumento del tempo necessario per la costruzione del modello da parte di *NuSMV*[1].

[1]Model checker utilizzato: `https://nusmv.fbk.eu/open_nusmv/flier.html`

Capitolo 2

Regole di transizione

La semantica utilizzata è quella di tipo astratto. All'interno di questa ogni valore concreto, composto da una coppia (valore, livello di sicurezza), viene approssimato considerando solo il suo livello di sicurezza. Come conseguenza, quando si tratta di comandi condizionali o iterativi, il sistema di transizione astratto ha percorsi di esecuzione multipli a causa della perdita di precisione dei dati astratti; nel caso dell'istruzione condizionale *IF* vengono applicate entrambe le regole dato che *true* e *false* sono entrambi astratti.
Gli stati sono definiti dalla tupla

$$< \sigma, PC, M, S, \rho >$$

- σ: environment.
- **PC**: program counter.
- **M**: memoria .
- **S**: operand stack $(\sigma_1, ..., \sigma_n)$. Utilizzando la semantica astratta all'interno dell'operand stack sono memorizzati unicamente i livelli di sicurezza σ_i.
- ρ: ipd stack (j, σ), ... (j', σ'). Mantiene le informazioni sui flussi impliciti aperti. Viene aggiornato ogni volta che viene inserita un'istruzione di controllo e ogni volta che un'istruzione di controllo termina. In particolare, j_i è l'indirizzo dove termina il flusso implicito i-esimo mentre σ_i è il livello di sicurezza dell'environment che deve essere ripristinato.

Le regole di transizione per le istruzioni implementate nel progetto sono le seguenti, al numeratore si ha il prefisso della regola mentre al denominatore si ha come cambia lo stato in seguito alla transizione (a sinistra della freccia si ha lo stato corrente mentre sulla destra lo stato successivo) :

- **load**:

$$\text{load}\ \frac{c[i] = load\, x \quad M[x] = (k, \tau) \quad not\_top(i, \rho)}{< \sigma, i, M, S, \rho > \rightarrow < \sigma, i+1, (k, \sigma \cup \tau) \cdot S, \rho >}$$

L'istruzione all'indirizzo i è una *load* e tale indirizzo non si trova nel top dell'ipd stack. All'indirizzo x in memoria si ha una coppia (valore k, livello di sicurezza τ). Lo stato successivo differisce da quello corrente solamente per l'operand stack. L'applicazione della regola consiste nel mettere in cima all'operand stack il valore k contenuto in memoria all'indirizzo x e assegnare come livello di sicurezza l'upper bound level tra l'environment σ dello stato corrente e il livello di sicurezza in x, τ. Utilizzando la semantica astratta il valore k non è conosciuto, dunque si utilizza unicamente il livello di sicurezza τ.

- **store**:

$$\text{store}\ \frac{c[i] = store \quad not\_top(i, \rho)}{< \sigma, i, M, (k, \tau) \cdot S, \rho > \rightarrow < \sigma, i+1, M[(k, \tau)/x], S, \rho >}$$

In questo caso si ha all'indirizzo i-esimo l'istruzione *store* la quale come argomento riceve la variabile x in memoria in cui sarà memorizzata la coppia (valore, livello di sicurezza) attualmente in cima all'operand stack. Anche in questo caso si ha come prerequisito che l'istruzione non si trovi in cima all'ipd stack. Applicando la regola verrà quindi rimossa la coppia dal top dell'operand stack la quale sarà memorizzata in memoria nella variabile x; il livello di sicurezza da assegnare alla variabile sarà ricalcolato come l'upper bound tra l'environment σ e il livello di sicurezza attualmente in cima all'operand stack τ. Utilizzando la semantica astratta non si ha l'informazione riguardante il valore k ma unicamente il livello di sicurezza.

- **ipd**:

$$\text{ipd}\ \frac{\rho = (i, \tau) \cdot \rho'}{< \sigma, i, M, S, \rho > \rightarrow < \tau, i, M, S, \rho' >}$$

Il prerequisito di questa regola è che l'istruzione i-esima si trovi in cima all'ipd stack. Il program counter dello stato corrente in un certo momento raggiungerà l'indirizzo; tale indirizzo in cima all'ipd stack è l'immediato post dominator di un flusso implicito aperto precedentemente, cioè si tratta della prima istruzione eseguita al termine di entrambi i rami dell'istruzione condizionale IF. Nel momento in cui il program counter raggiunge dunque tale indirizzo i è necessario ripristinare l'environment che si aveva prima dell'apertura del flusso implicito. Lo stato successivo applicando questa regola avrà quindi come environment il livello di sicurezza τ salvato nell'ipd stack al momento dell'inizo del flusso inplicito, invece la coppia (ipd, livello di sicurezza) verrà rimossa dalla cima dell'ipd stack.

- **goto**:

$$\text{goto}\ \frac{c[i] = goto\, j \quad not\_top(i, \rho)}{< \sigma, i, M, S, \rho > \rightarrow < \sigma, j, M, S, \rho >}$$

L'istruzione *goto* consiste nel saltare all'istruzione che si trova all'indirizzo j-esimo passato come argomento. Applicando dunque tale regola lo stato successivo è dunque determinato semplicemente aggiornando il valore del program counter a j.

- **if-false**:

$$\text{if}_{\text{false}} \frac{c[i] = if\ j \quad not\_top(i, \rho)}{< \sigma, i, M, (0, \tau) \cdot S, \rho > \rightarrow < \tau, i+1, up_{\text{M}}(M, mod^P(F^P(i)), \tau), up_{\text{s}}(S, \tau), (ipd(i), \sigma) \cdot \rho >}$$

La seguente regola è applicata nel momento in cui all'indirizzo i-esimo si ha un'istruzione condizionale IF e in cima all'operand stack si ha all'interno della coppia (valore k, livello di sicurezza τ) un valore k uguale a 0. Il Program Counter dello stato successivo sarà semplicemente quello dello stato precedente incrementato di uno poiché in cima all'operand stack, come detto, si ha un valore nullo e dunque non è rispettata la condizione di salto. Avendo un istruzione condizionale IF si ha l'apertura di un flusso implicito, dunque l'environment dello stato successivo sarà uguale all'upper bound tra l'environment attuale e il livello di sicurezza rimosso dalla cima dell'operand stack; l'ipd nello stato successivo avrà memorizzato sul top l'ipd e il livello dell'environment dello stato precedente. Come già detto quando il PC raggiungerà il valore dell'ipd la coppia memorizzata in top all'ipd stack sarà rimossa e verrà ripristinato il valore dell'environment al valore memorizzato sul top dell'ipd stack.

- **if-true**:

$$\text{if}_{\text{true}} \frac{c[i] = if\ j \quad not\_top(i, \rho)}{< \sigma, i, M, (k \neq 0, \tau) \cdot S, \rho > \rightarrow < \tau, j, up_{\text{M}}(M, mod^P(F^P(i)), \tau), up_{\text{s}}(S, \tau), (ipd(i), \sigma) \cdot \rho >}$$

Tale regola viene applicata quando la condizione di salto dell'istruzione condizionale IF è rispettata dunque quando in cima all'operand stack si ha un valore diverso da 0. Anche in questo caso come nel caso *false* viene rimossa la coppia memorizzata in cima all'operand stack e viene aperto un flusso implicito, dunque è necessario aggiornare il livello dell'environment nello stato successivo e memorizzare la coppia (ipd, livello di sicurezza da ripristinare) in cima all'ipd stack.

Capitolo 3

Scelte di design e implementazione

3.1 Assunzioni iniziali

Come accennato nel capitolo introduttivo, l'elevato numero di stati possibili che viene a crearsi al crescere della dimensione del set di istruzioni possibili e il numero di istruzioni che costituiscono il programma ci ha portato ad adottare alcune semplificazioni. Nello specifico:

- Il set di istruzioni bytecode gestite dal model checker sono quelle di `load`, `store`, `push`, `pop`, `goto`, `if`, `halt`.
- Il nostro design prevede che siano disponibili solamente tre locazioni di memoria; i valori che questi possono assumere (in semantica astratta) sono `lo`, `hi` oppure `null` (non inizializzato).
- Per quanto riguarda lo stack degli operandi, abbiamo previsto 5 posizioni diverse assegnabili nel corso di una singola esecuzione di un programma.
- Analogamente, abbiamo previsto 2 locazioni di memoria assegnabili per lo stack dell'*ipd*.

3.2 Struttura del modello

Il file .smv del modello è sviluppato su più moduli, ognuno dei quali implementa un diverso elemento che costituisce lo stato del programma

3.2.1 `global`

Il modulo `global` contiene delle variabili globali, utilizzate da più moduli del model checker. Nello specifico, la variabile `inst_upper_bound` è necessaria per tutte le strutture dati che dipendono direttamente dal numero di istruzioni. L'utilizzo di una variabile globale permette di avere una maggiore flessibilità, qualora fosse necessario ricompilare il modello per un numero di istruzioni diverso; in tal caso, basterà infatti modificare soltanto il valore di questa variabile, un'unica volta.

3.2.2 instructions

Questo modulo contiene tutte le istruzioni, insieme al rispettivo argomento e ipd, che costituiscono il programma che si desidera analizzare.

È costituito, nello specifico, da tre array di uguale dimensione (e pari al numero di istruzioni di cui il programma è composto), uno per ciascuno dei parametri che fanno variare lo stato. Nel caso in cui un'istruzione non richieda un argomento esplicitamente, o per cui non sia necessario specificare l'ipd di un'istruzione, si è adottata la convenzione di utilizzare un valore numerico di 0, che viene ignorato dal model checker.

Le variabili sono di tipo FROZENVAR, non dovendo mutare nel corso dell'esecuzione del programma. Inoltre, tutte le variabili di questo modulo sono inizializzate dallo script python (paragrafo 3.3).

3.2.3 op_stack

Questo modulo contiene la struttura dati dell'operand stack del programma. È costituito da un array di 5 posizioni che possono assumere i valori `lo`, `hi` e `null`. Tali valori corrispondono, rispettivamente, ad un livello di sicurezza basso, alto e ad un valore non inizializzato. L'introduzione di un tipo `null` si è rivelato estremamente utile in fase di sviluppo a fini di debug, e permette di seguire senza ambiguità l'evoluzione degli stati nel transition system.

Una variabile intera `top` che assume valori nel range $[-1, 4]$ tiene traccia, in ogni momento, dell'indice dell'ultima posizione occupata all'interno dell'array (ToS).

Il valore di top è inizializzato a -1, per indicare che inizialmente lo stack degli operandi è vuoto, e ciascuna variabile all'interno dell'array dei valori è settato a `null`.

3.2.4 ipd_stack

Il modulo `ipd_stack` ha una struttura del tutto analoga a quella di `op_stack`. La differenza in questo caso è che l'array è di sole due posizioni[1], ed è presente un secondo array di variabili, `addr` che contiene il numero di istruzione dopo il quale il valore dell'ambiente prima di entrare nel flusso implicito deve essere ripristinato.

Come per lo stack degli operandi, anche in questo caso un valore di tipo `null` indica una locazione di memoria vuota o non inizializzata.

3.2.5 state

Il modulo `state` è senza dubbio il più importante del model checker. Oltre ad essere un wrapper per l'istanziazione dei precedenti moduli, in esso sono presenti tutte le regole di transizioni delle varie variabili.

In aggiunta alle variabili descritte nelle precedenti sezioni, sono presenti quattro ulteriori variabili, necessarie a descrivere lo stato del transition system:

- `pc`: Questa variabile rappresenta il program counter del programma; inizializzato a 0, contiene sempre il numero della successiva istruzione da eseguire. Dopo ogni istruzione viene incrementato di uno, a meno che la precedente istruzione non

[1] Tale semplificazione è nuovamente dovuta al problema dell'aumento esponenziale del numero di stati. Non dovendo analizzare, nel nostro caso, programmi eccessivamente lunghi, abbiamo ritenuto sufficiente assegnare due posizioni all'array, così da limitare il numero degli stati da esplorare.

fosse una `goto` o `if`, e se il programma non è terminato (variabile `halted` settata a `true`).

- `memory`: Un array di tre posizioni che rappresenta la memoria di cui può fare uso il programma. I valori che può assumere sono `hi`, `lo` e `null` e sono inizializzati dallo script Python.
- `env` rappresenta, in ogni istante, il livello attuale di sicurezza del programma.
- `halted` è un semplice flag che indica lo stato attuale di esecuzione (o non-esecuzione) del programma. Non appena viene incontrata un'istruzione di tipo `halt` il flag viene settato a `true` e il transition system smette di evolvere.

3.2.6 `main`

L'ultimo modulo del modello è `main`. In questo modulo sono presenti un'istanziazione del modulo `state` ed è definita la proprietà del secure information flow del programma che deve essere verificato. In particolare, la condizione verifica che in ogni possibile stato finale, il livello di sicurezza di ciascuna variabile in memoria che aveva un valore iniziale `lo` non dipende dallo stato iniziale di una variabile che possiede un valore di sicurezza più alto. La proprietà viene riadattata e riscritta in base al modello che si vuole analizzare dallo script python, dopo che sono stati decisi i valori iniziali per la memoria.

3.3 Script Python per la generazione del modello NuSMV

Per rendere più rapida l'inizializzazione delle variabili necessarie a testare le proprietà del secure information flow con diversi programmi, abbiamo realizzato uno script python, che riscrive alcune sezioni del codice .smv.

Per eseguire lo script è sufficiente lanciare `./model_checker_generator.py` (Python 3.6 o superiore richiesto). Lo script può essere lanciato in due modalità:

- La prima, senza specificare alcuna seconda opzione, è *interattiva*. Inizialmente viene richiesto il numero totale di istruzioni che costituirà il programma, e successivamente, per ciascuna di queste, vengono richiesti istruzione, argomento e ipd riferito all'istruzione corrente. Infine, vengono richiesti i livelli di sicurezza delle tre locazioni di memoria consentite. Al termine dell'esecuzione, vengono generati un file di testo "bytecode", contenente tutte le informazioni necessarie per poter ricreare più rapidamente lo stesso modello in un secondo momento (istruzioni nella forma `instr arg ipd`) e il file "generated_model.smv", contentente il modello NuSMV vero e proprio.
- La seconda possibilità è quella di passare il file bytecode come argomento allo script. In questo caso, lo script effettuerà un parsing direttamente dal file per recuperare le istruzioni, e verranno richiesti soltanto i valori iniziali delle locazioni di memoria.

Capitolo 4

Casi Analizzati

In questa sezione sono riportati i casi analizzati con NuSMV per verificare quali di essi hanno un flusso di informazioni sicuro e quali no; in particolare per ciascuno di essi si è controllato che le variabili con un livello di sicurezza "LOW" non acquisiscano un livello di sicurezza "HIGH" in nessuno stato causando quindi un data leakage.

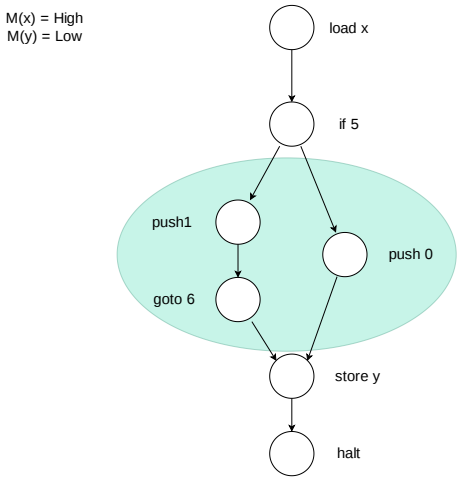
4.1 Primo caso

4.1.1 Bytecode

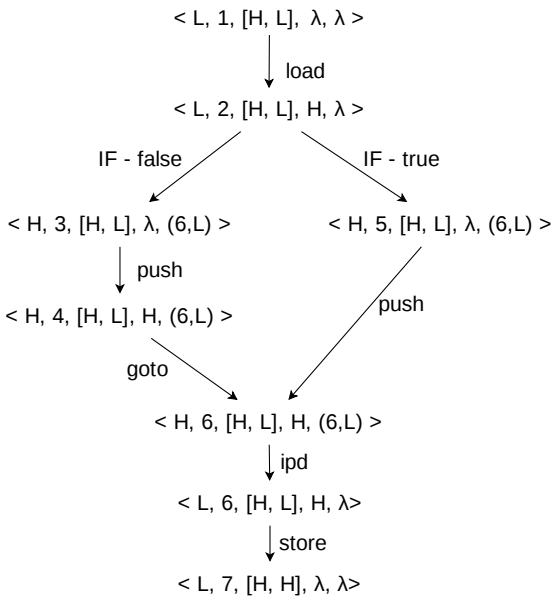
Qui di seguito è riportato il bytecode del primo caso analizzato.

```
1 load x
2 if 5
3 push 1
4 goto 6
5 push 0
6 store y
7 halt
```

4.1.2 Control flow graph



4.1.3 Analisi



Dall'analisi delle transizioni da uno stato all'altro, effettuate tendendo conto delle regole elencate precedentemente, è possibile notare che tale bytecode non è un flusso sicuro di informazioni poichè quando termina, la variabile y, inizialmente inizializzata con un livello di sicurezza basso, ha acquisito un livello di sicurezza alto. Questo è quindi un caso di data leakage.
Qui di seguito è riportato, stato per stato, l'analisi effettuata mediante l'utilizzo del model checker NuSMV. La costruzione del modello (ossia del file .smv) è effettuata mediante l'utilizzo dello script python[1] contenuto nella directory in cui è inclusa la seguente documentazione.

1. `./NuSMV -int`
   attiva una shell interattiva

2. `read_model -i generated_model.smv`
   legge l'input model

3. `go`
   legge e inizializza NuSMV per la verifica o la simulazione

4. `pick_state -v -r`
   Viene selezionato uno stato iniziale in maniera casuale tra tutti gli stati iniziali. In questo caso ne abbiamo solamente uno

   Output:

```
1     Trace Description: Simulation Trace
2     Trace Type: Simulation
3       -> State: 1.1 <-
4         instructions.inst[0] = load
5         instructions.inst[1] = if
```

[1]si tratta dello script: model_checker_generator.py

```
6           instructions.inst[2] = push
7           instructions.inst[3] = goto
8           instructions.inst[4] = push
9           instructions.inst[5] = store
10          instructions.inst[6] = halt
11          instructions.arg[0] = 0
12          instructions.arg[1] = 4
13          instructions.arg[2] = 0
14          instructions.arg[3] = 5
15          instructions.arg[4] = 0
16          instructions.arg[5] = 1
17          instructions.arg[6] = 0
18          instructions.ipd[0] = 0
19          instructions.ipd[1] = 5
20          instructions.ipd[2] = 0
21          instructions.ipd[3] = 0
22          instructions.ipd[4] = 0
23          instructions.ipd[5] = 0
24          instructions.ipd[6] = 0
25          state.pc = 0
26          state.memory[0] = hi
27          state.memory[1] = lo
28          state.memory[2] = null
29          state.ipd_s.val[0] = null
30          state.ipd_s.val[1] = null
31          state.ipd_s.addr[0] = 0
32          state.ipd_s.addr[1] = 0
33          state.ipd_s.top = -1
34          state.op_s.val[0] = null
35          state.op_s.val[1] = null
36          state.op_s.val[2] = null
37          state.op_s.val[3] = null
38          state.op_s.val[4] = null
39          state.op_s.top = -1
40          state.env = lo
41          state.halted = FALSE
42          global.inst_upper_bound = 6
43          global.n_inst = 7
44          instructions.global.inst_upper_bound = 6
45          instructions.global.n_inst = 7
46          state.global.inst_upper_bound = 6
47          state.global.n_inst = 7
48          state.ipd_s.global.inst_upper_bound = 6
49          state.ipd_s.global.n_inst = 7
```

Dall'outuput è possibile vedere che tutte le strutture dati necessarie all'analisi sono state inizializzate correttamente.

5. `simulate -p -i`
   Genera una sequenza di stati partendo dallo stato corrente e ad ogni step sceglie lo stato successivo in maniera interattiva. Mediante l'opzione -p è possibile stampare

unicamente le variabili di stato di cui è modificato il valore.
Output:

```
******** Simulation Starting From State 1.1 ********
*************** AVAILABLE STATES *************
  ================ State ================
  0) ------------------------
  global.inst_upper_bound = 6
  global.n_inst = 7
  instructions.global.inst_upper_bound = 6
  instructions.global.n_inst = 7
  instructions.inst[0] = load
  instructions.inst[1] = if
  instructions.inst[2] = push
  instructions.inst[3] = goto
  instructions.inst[4] = push
  instructions.inst[5] = store
  instructions.inst[6] = halt
  instructions.arg[0] = 0
  instructions.arg[1] = 4
  instructions.arg[2] = 0
  instructions.arg[3] = 5
  instructions.arg[4] = 0
  instructions.arg[5] = 1
  instructions.arg[6] = 0
  instructions.ipd[0] = 0
  instructions.ipd[1] = 5
  instructions.ipd[2] = 0
  instructions.ipd[3] = 0
  instructions.ipd[4] = 0
  instructions.ipd[5] = 0
  instructions.ipd[6] = 0
  state.global.inst_upper_bound = 6
  state.global.n_inst = 7
  state.pc = 1
  state.memory[0] = hi
  state.memory[1] = lo
  state.memory[2] = null
  state.ipd_s.global.inst_upper_bound = 6
  state.ipd_s.global.n_inst = 7
  state.ipd_s.val[0] = null
  state.ipd_s.val[1] = null
  state.ipd_s.addr[0] = 0
  state.ipd_s.addr[1] = 0
  state.ipd_s.top = -1
  state.op_s.val[0] = hi
  state.op_s.val[1] = null
  state.op_s.val[2] = null
  state.op_s.val[3] = null
  state.op_s.val[4] = null
  state.op_s.top = 0
  state.env = lo
```

```
50        state.halted = FALSE
51      There's only one available state. Press Return to Proceed.
```

Si può notare che è stata correttamente eseguita la prima istruzione di load, infatti il valore in prima posizione nell'operand stack è stato correttamente settato ad high.

6. Premendo Invio si seleziona l'unico stato disponibile dopo quello corrente.
   Output:

```
1       Chosen state is: 0
2       ***************  AVAILABLE STATES  *************
3             ================= State =================
4             0) ------------------------
5             global.inst_upper_bound = 6
6             global.n_inst = 7
7             instructions.global.inst_upper_bound = 6
8             instructions.global.n_inst = 7
9             instructions.inst[0] = load
10            instructions.inst[1] = if
11            instructions.inst[2] = push
12            instructions.inst[3] = goto
13            instructions.inst[4] = push
14            instructions.inst[5] = store
15            instructions.inst[6] = halt
16            instructions.arg[0] = 0
17            instructions.arg[1] = 4
18            instructions.arg[2] = 0
19            instructions.arg[3] = 5
20            instructions.arg[4] = 0
21            instructions.arg[5] = 1
22            instructions.arg[6] = 0
23            instructions.ipd[0] = 0
24            instructions.ipd[1] = 5
25            instructions.ipd[2] = 0
26            instructions.ipd[3] = 0
27            instructions.ipd[4] = 0
28            instructions.ipd[5] = 0
29            instructions.ipd[6] = 0
30            state.global.inst_upper_bound = 6
31            state.global.n_inst = 7
32            state.pc = 4
33            state.memory[0] = hi
34            state.memory[1] = lo
35            state.memory[2] = null
36            state.ipd_s.global.inst_upper_bound = 6
37            state.ipd_s.global.n_inst = 7
38            state.ipd_s.val[0] = lo
39            state.ipd_s.val[1] = null
40            state.ipd_s.addr[0] = 5
41            state.ipd_s.addr[1] = 0
42            state.ipd_s.top = 0
```

```
43            state.op_s.val[0] = null
44            state.op_s.val[1] = null
45            state.op_s.val[2] = null
46            state.op_s.val[3] = null
47            state.op_s.val[4] = null
48            state.op_s.top = -1
49            state.env = hi
50            state.halted = FALSE
51
52
53            ================ State ================
54            1) ------------------------
55            state.pc = 2
56      Choose a state from the above (0-1):
```

Dopo aver premuto invio si può notare che è stato aperto correttamente il flusso implicito. A questo punto si deve selezionare quale dei due rami dell'if percorrere selezionando uno dei due stati. In entrambi avremo l'environment settato al livello high.

7. Si selezioni adesso 0. In questo modo si sceglie di percorrere il ramo true dell'IF. **Output:**

```
1       Chosen state is: 0
2       ***************  AVAILABLE STATES  *************
3         ================= State =================
4         0) ------------------------
5         global.inst_upper_bound = 6
6         global.n_inst = 7
7         instructions.global.inst_upper_bound = 6
8         instructions.global.n_inst = 7
9         instructions.inst[0] = load
10        instructions.inst[1] = if
11        instructions.inst[2] = push
12        instructions.inst[3] = goto
13        instructions.inst[4] = push
14        instructions.inst[5] = store
15        instructions.inst[6] = halt
16        instructions.arg[0] = 0
17        instructions.arg[1] = 4
18        instructions.arg[2] = 0
19        instructions.arg[3] = 5
20        instructions.arg[4] = 0
21        instructions.arg[5] = 1
22        instructions.arg[6] = 0
23        instructions.ipd[0] = 0
24        instructions.ipd[1] = 5
25        instructions.ipd[2] = 0
26        instructions.ipd[3] = 0
27        instructions.ipd[4] = 0
28        instructions.ipd[5] = 0
29        instructions.ipd[6] = 0
```

```
30        state.global.inst_upper_bound = 6
31        state.global.n_inst = 7
32        state.pc = 5
33        state.memory[0] = hi
34        state.memory[1] = lo
35        state.memory[2] = null
36        state.ipd_s.global.inst_upper_bound = 6
37        state.ipd_s.global.n_inst = 7
38        state.ipd_s.val[0] = lo
39        state.ipd_s.val[1] = null
40        state.ipd_s.addr[0] = 5
41        state.ipd_s.addr[1] = 0
42        state.ipd_s.top = 0
43        state.op_s.val[0] = hi
44        state.op_s.val[1] = null
45        state.op_s.val[2] = null
46        state.op_s.val[3] = null
47        state.op_s.val[4] = null
48        state.op_s.top = 0
49        state.env = hi
50        state.halted = FALSE
51
52      There's only one available state. Press Return to Proceed.
```

È possibile notare in questo caso che è stata eseguita con successo anche l'operazione di push. Infatti il valore in cima all'operand stack è stato settato correttamente ad high.

8. A questo punto è possibile premere invio selezionando l'unico stato successivo disponibile. In questo modo verrà eseguita la regola di transizione relativa all'istruzione *ipd*. Se eseguita correttamente verrà ripristinato il valore dell'environment a low (valore nel momento in cui si è aperto il flusso implicito) e verrà rimossa la coppia (ipd, environment) dal top dell'ipd stack. È possibile riscontrare che ciò viene fatto dall'output:
   Output:

```
1       Chosen state is: 0
2       ***************  AVAILABLE STATES  *************
3         ================= State =================
4         0) ------------------------
5         global.inst_upper_bound = 6
6         global.n_inst = 7
7         instructions.global.inst_upper_bound = 6
8         instructions.global.n_inst = 7
9         instructions.inst[0] = load
10        instructions.inst[1] = if
11        instructions.inst[2] = push
12        instructions.inst[3] = goto
13        instructions.inst[4] = push
14        instructions.inst[5] = store
15        instructions.inst[6] = halt
16        instructions.arg[0] = 0
```

```
17        instructions.arg[1] = 4
18        instructions.arg[2] = 0
19        instructions.arg[3] = 5
20        instructions.arg[4] = 0
21        instructions.arg[5] = 1
22        instructions.arg[6] = 0
23        instructions.ipd[0] = 0
24        instructions.ipd[1] = 5
25        instructions.ipd[2] = 0
26        instructions.ipd[3] = 0
27        instructions.ipd[4] = 0
28        instructions.ipd[5] = 0
29        instructions.ipd[6] = 0
30        state.global.inst_upper_bound = 6
31        state.global.n_inst = 7
32        state.pc = 5
33        state.memory[0] = hi
34        state.memory[1] = lo
35        state.memory[2] = null
36        state.ipd_s.global.inst_upper_bound = 6
37        state.ipd_s.global.n_inst = 7
38        state.ipd_s.val[0] = null
39        state.ipd_s.val[1] = null
40        state.ipd_s.addr[0] = 0
41        state.ipd_s.addr[1] = 0
42        state.ipd_s.top = -1
43        state.op_s.val[0] = hi
44        state.op_s.val[1] = null
45        state.op_s.val[2] = null
46        state.op_s.val[3] = null
47        state.op_s.val[4] = null
48        state.op_s.top = 0
49        state.env = lo
50        state.halted = FALSE
51
52
53    There's only one available state. Press Return to Proceed.
```

9. A questo si prosegue l'analisi. Premendo invio si accede all'unico e ultimo stato accessibile.
   Output:

```
1     Chosen state is: 0
2     ***************  AVAILABLE STATES  *************
3
4       ================= State =================
5       0) -------------------------
6       global.inst_upper_bound = 6
7       global.n_inst = 7
8       instructions.global.inst_upper_bound = 6
9       instructions.global.n_inst = 7
10      instructions.inst[0] = load
```

```
11        instructions.inst[1] = if
12        instructions.inst[2] = push
13        instructions.inst[3] = goto
14        instructions.inst[4] = push
15        instructions.inst[5] = store
16        instructions.inst[6] = halt
17        instructions.arg[0] = 0
18        instructions.arg[1] = 4
19        instructions.arg[2] = 0
20        instructions.arg[3] = 5
21        instructions.arg[4] = 0
22        instructions.arg[5] = 1
23        instructions.arg[6] = 0
24        instructions.ipd[0] = 0
25        instructions.ipd[1] = 5
26        instructions.ipd[2] = 0
27        instructions.ipd[3] = 0
28        instructions.ipd[4] = 0
29        instructions.ipd[5] = 0
30        instructions.ipd[6] = 0
31        state.global.inst_upper_bound = 6
32        state.global.n_inst = 7
33        state.pc = 6
34        state.memory[0] = hi
35        state.memory[1] = hi
36        state.memory[2] = null
37        state.ipd_s.global.inst_upper_bound = 6
38        state.ipd_s.global.n_inst = 7
39        state.ipd_s.val[0] = null
40        state.ipd_s.val[1] = null
41        state.ipd_s.addr[0] = 0
42        state.ipd_s.addr[1] = 0
43        state.ipd_s.top = -1
44        state.op_s.val[0] = null
45        state.op_s.val[1] = null
46        state.op_s.val[2] = null
47        state.op_s.val[3] = null
48        state.op_s.val[4] = null
49        state.op_s.top = -1
50        state.env = lo
51        state.halted = FALSE
```

È possibile notare che è stata eseguita con successo l'ultima istruzione, la store. Questa rimuove il livello di sicurezza presente sulla cima dell'operand stack e lo assegna alla variabile y in memoria. Si è quindi dimostrato che tale flusso non è sicuro in quanto la variabile y, inizialmente con un livello di sicurezza low, quando termina l'esecuzione del bytecode ha un livello di sicurezza superiore high.

10. Tornando al punto 6, ecco l'ouput corrispondente alla scelta dello state 1 corrispondende al percorrimento del ramo FALSE dell'istruzione condizionale.
Output:

```
Chosen state is: 1
*************** AVAILABLE STATES  *************
  ================= State =================
  0) -------------------------
  global.inst_upper_bound = 6
  global.n_inst = 7
  instructions.global.inst_upper_bound = 6
  instructions.global.n_inst = 7
  instructions.inst[0] = load
  instructions.inst[1] = if
  instructions.inst[2] = push
  instructions.inst[3] = goto
  instructions.inst[4] = push
  instructions.inst[5] = store
  instructions.inst[6] = halt
  instructions.arg[0] = 0
  instructions.arg[1] = 4
  instructions.arg[2] = 0
  instructions.arg[3] = 5
  instructions.arg[4] = 0
  instructions.arg[5] = 1
  instructions.arg[6] = 0
  instructions.ipd[0] = 0
  instructions.ipd[1] = 5
  instructions.ipd[2] = 0
  instructions.ipd[3] = 0
  instructions.ipd[4] = 0
  instructions.ipd[5] = 0
  instructions.ipd[6] = 0
  state.global.inst_upper_bound = 6
  state.global.n_inst = 7
  state.pc = 3
  state.memory[0] = hi
  state.memory[1] = lo
  state.memory[2] = null
  state.ipd_s.global.inst_upper_bound = 6
  state.ipd_s.global.n_inst = 7
  state.ipd_s.val[0] = lo
  state.ipd_s.val[1] = null
  state.ipd_s.addr[0] = 5
  state.ipd_s.addr[1] = 0
  state.ipd_s.top = 0
  state.op_s.val[0] = hi
  state.op_s.val[1] = null
  state.op_s.val[2] = null
  state.op_s.val[3] = null
  state.op_s.val[4] = null
  state.op_s.top = 0
  state.env = hi
  state.halted = FALSE
```

Come è possibile osservare in seguito alla push eseguita all'interno del branch

false viene inserito in cima all'operand stack il livello di sicurezza high, uguale al valore dell'environment corrente.

11. Si prosegue premendo invio ed esplorando l'unico stato disponibile il quale è lo stato raggiunto in seguito all'esecuzione dell'istruzione *goto 6*. In questo modo si salterà direttamente all'istruzione all'indirizzo 6, ossia *store y*. Prima di eseguirla verrà eseguita l'istruzione *ipd*.

```
Chosen state is: 0
**************  AVAILABLE STATES  *************
  ================ State ================
  0) ------------------------
  global.inst_upper_bound = 6
  global.n_inst = 7
  instructions.global.inst_upper_bound = 6
  instructions.global.n_inst = 7
  instructions.inst[0] = load
  instructions.inst[1] = if
  instructions.inst[2] = push
  instructions.inst[3] = goto
  instructions.inst[4] = push
  instructions.inst[5] = store
  instructions.inst[6] = halt
  instructions.arg[0] = 0
  instructions.arg[1] = 4
  instructions.arg[2] = 0
  instructions.arg[3] = 5
  instructions.arg[4] = 0
  instructions.arg[5] = 1
  instructions.arg[6] = 0
  instructions.ipd[0] = 0
  instructions.ipd[1] = 5
  instructions.ipd[2] = 0
  instructions.ipd[3] = 0
  instructions.ipd[4] = 0
  instructions.ipd[5] = 0
  instructions.ipd[6] = 0
  state.global.inst_upper_bound = 6
  state.global.n_inst = 7
  state.pc = 5
  state.memory[0] = hi
  state.memory[1] = lo
  state.memory[2] = null
  state.ipd_s.global.inst_upper_bound = 6
  state.ipd_s.global.n_inst = 7
  state.ipd_s.val[0] = null
  state.ipd_s.val[1] = null
  state.ipd_s.addr[0] = 0
  state.ipd_s.addr[1] = 0
  state.ipd_s.top = -1
  state.op_s.val[0] = hi
  state.op_s.val[1] = null
```

```
45        state.op_s.val[2] = null
46        state.op_s.val[3] = null
47        state.op_s.val[4] = null
48        state.op_s.top = 0
49        state.env = lo
50        state.halted = FALSE
```

L'environment è stato ripristinato correttamente ed è anche stato rimosso il contenuto presente sulla cima dell'ipd stack.

12. Premendo invio viene eseguito l'unico ed ultimo stato, raggiunto mediante l'esecuzione della store y. Anche in questo caso si avrà un data leakage, poichè verrà nuovamente assegnato un livello di sicurezza high alla variabile y, inizialmente impostata ad un livello di sicurezza basso.

```
1     Chosen state is: 0
2     ***************  AVAILABLE STATES  *************
3       ================= State =================
4       0) ------------------------
5       global.inst_upper_bound = 6
6       global.n_inst = 7
7       instructions.global.inst_upper_bound = 6
8       instructions.global.n_inst = 7
9       instructions.inst[0] = load
10      instructions.inst[1] = if
11      instructions.inst[2] = push
12      instructions.inst[3] = goto
13      instructions.inst[4] = push
14      instructions.inst[5] = store
15      instructions.inst[6] = halt
16      instructions.arg[0] = 0
17      instructions.arg[1] = 4
18      instructions.arg[2] = 0
19      instructions.arg[3] = 5
20      instructions.arg[4] = 0
21      instructions.arg[5] = 1
22      instructions.arg[6] = 0
23      instructions.ipd[0] = 0
24      instructions.ipd[1] = 5
25      instructions.ipd[2] = 0
26      instructions.ipd[3] = 0
27      instructions.ipd[4] = 0
28      instructions.ipd[5] = 0
29      instructions.ipd[6] = 0
30      state.global.inst_upper_bound = 6
31      state.global.n_inst = 7
32      state.pc = 6
33      state.memory[0] = hi
34      state.memory[1] = hi
35      state.memory[2] = null
36      state.ipd_s.global.inst_upper_bound = 6
37      state.ipd_s.global.n_inst = 7
```

```
38        state.ipd_s.val[0] = null
39        state.ipd_s.val[1] = null
40        state.ipd_s.addr[0] = 0
41        state.ipd_s.addr[1] = 0
42        state.ipd_s.top = -1
43        state.op_s.val[0] = null
44        state.op_s.val[1] = null
45        state.op_s.val[2] = null
46        state.op_s.val[3] = null
47        state.op_s.val[4] = null
48        state.op_s.top = -1
49        state.env = lo
50        state.halted = FALSE
```

Dall'analisi interattiva effettuata mediante l'utilizzo del model checker NuSMV è stato dunque individuato un data leakage dovuto all'assegnamento di un livello di sicurezza alto ad una variabile il cui livello di sicurezza iniziale era stato impostato come basso. Ciò accade sia che venga seguito il ramo true che quello false dell'istruzione condizionale IF.
Invocando il model checker in questo modo: `./NuSMV generated_model.smv` è possibile effettuare il check automatico sul modello così che il model checker mostri direttamente se è soddisfatta o meno la formula logica impostata all'interno del file .smv. In questo caso la formula logica è la seguente:

$$AG\ state.memory[1] = lo$$

In questo modo si richiede che lungo tutti i possibili percorsi valga in ogni stato che la variabile y, salvata nella seconda posizione libera in memoria, abbia sempre un livello di sicurezza basso.
Output:

```
1  -- specification AG state.memory[1] = lo  is false
2  -- as demonstrated by the following execution sequence
3  Trace Description: CTL Counterexample
4  Trace Type: Counterexample
5    -> State: 1.1 <-
6      instructions.inst[0] = load
7      instructions.inst[1] = if
8      instructions.inst[2] = push
9      instructions.inst[3] = goto
10     instructions.inst[4] = push
11     instructions.inst[5] = store
12     instructions.inst[6] = halt
13     instructions.arg[0] = 0
14     instructions.arg[1] = 4
15     instructions.arg[2] = 0
16     instructions.arg[3] = 5
17     instructions.arg[4] = 0
18     instructions.arg[5] = 1
19     instructions.arg[6] = 0
20     instructions.ipd[0] = 0
```

```
21      instructions.ipd[1] = 5
22      instructions.ipd[2] = 0
23      instructions.ipd[3] = 0
24      instructions.ipd[4] = 0
25      instructions.ipd[5] = 0
26      instructions.ipd[6] = 0
27      state.pc = 0
28      state.memory[0] = hi
29      state.memory[1] = lo
30      state.memory[2] = null
31      state.ipd_s.val[0] = null
32      state.ipd_s.val[1] = null
33      state.ipd_s.addr[0] = 0
34      state.ipd_s.addr[1] = 0
35      state.ipd_s.top = -1
36      state.op_s.val[0] = null
37      state.op_s.val[1] = null
38      state.op_s.val[2] = null
39      state.op_s.val[3] = null
40      state.op_s.val[4] = null
41      state.op_s.top = -1
42      state.env = lo
43      state.halted = FALSE
44      global.inst_upper_bound = 6
45      global.n_inst = 7
46      instructions.global.inst_upper_bound = 6
47      instructions.global.n_inst = 7
48      state.global.inst_upper_bound = 6
49      state.global.n_inst = 7
50      state.ipd_s.global.inst_upper_bound = 6
51      state.ipd_s.global.n_inst = 7
52    -> State: 1.2 <-
53      state.pc = 1
54      state.op_s.val[0] = hi
55      state.op_s.top = 0
56    -> State: 1.3 <-
57      state.pc = 4
58      state.ipd_s.val[0] = lo
59      state.ipd_s.addr[0] = 5
60      state.ipd_s.top = 0
61      state.op_s.val[0] = null
62      state.op_s.top = -1
63      state.env = hi
64    -> State: 1.4 <-
65      state.pc = 5
66      state.op_s.val[0] = hi
67      state.op_s.top = 0
68    -> State: 1.5 <-
69      state.ipd_s.val[0] = null
70      state.ipd_s.addr[0] = 0
71      state.ipd_s.top = -1
```

```
72      state.env = lo
73    -> State: 1.6 <-
74      state.pc = 6
75      state.memory[1] = hi
76      state.op_s.val[0] = null
77      state.op_s.top = -1
```

Come osservabile dall'output è confermato che il flusso di sicurezza non è sicuro e viene anche mostrato una controesempio che dimistra che la variabile y assume un livello di sicurezza alto.

4.2 Secondo caso

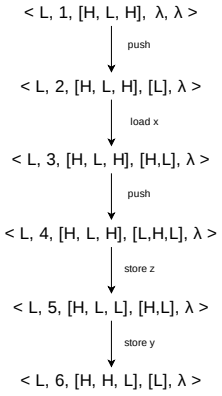
4.2.1 Bytecode

Qui di seguito è riportato il bytecode del primo caso analizzato.

```
1 push 0
2 load x
3 push 0
4 store z
5 store y
6 halt 0
```

in cui supponiamo: M(x) = High, M(y) = Low, M(z)=High e che in memoria la variabile x sia nella locazione con indirizzo 0, la variabile y in quella con indirizzo 1 e la variabile z in quella con indirizzo 2.

4.2.2 Analisi



Applicando nuovamente le regole elencate precedentemente nell'analisi delle transizioni, notiamo che il bytecode risulta essere un flusso insicuro di informazioni poiché alla terminazione la variabile Y, definita all'inizio con un livello di sicurezza basso, acquista un livello alto in quanto l'ordine di inserimento e rimozione dei dati in cima allo stack non avviene in modo sicuro.
Di seguito è riportata, stato per stato, l'analisi effettuata con NuSMV. Invocando il model checker con il comando `./NuSMV generrated_model.smv` esso dimostra che che la stessa formula logica dell'altro caso non è soddisfatta. Per brevità vengono mostrate soltanto le variabili di cui si è modificato il valore.

Output:

```
-- specification AG state.memory[1] = lo  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    instructions.inst[0] = push
    instructions.inst[1] = load
    instructions.inst[2] = push
    instructions.inst[3] = store
    instructions.inst[4] = store
    instructions.inst[5] = halt
    instructions.arg[0] = 0
    instructions.arg[1] = 0
    instructions.arg[2] = 0
    instructions.arg[3] = 2
    instructions.arg[4] = 1
    instructions.arg[5] = 0
    instructions.ipd[0] = 0
    instructions.ipd[1] = 0
    instructions.ipd[2] = 0
    instructions.ipd[3] = 0
    instructions.ipd[4] = 0
    instructions.ipd[5] = 0
    state.pc = 0
    state.memory[0] = hi
    state.memory[1] = lo
    state.memory[2] = hi
    state.ipd_s.val[0] = null
    state.ipd_s.val[1] = null
    state.ipd_s.addr[0] = 0
    state.ipd_s.addr[1] = 0
    state.ipd_s.top = -1
    state.op_s.val[0] = null
    state.op_s.val[1] = null
    state.op_s.val[2] = null
    state.op_s.val[3] = null
    state.op_s.val[4] = null
    state.op_s.top = -1
    state.env = lo
    state.halted = FALSE
    global.inst_upper_bound = 5
    global.n_inst = 6
    instructions.global.inst_upper_bound = 5
    instructions.global.n_inst = 6
    state.global.inst_upper_bound = 5
    state.global.n_inst = 6
    state.ipd_s.global.inst_upper_bound = 5
    state.ipd_s.global.n_inst = 6
  -> State: 1.2 <-
    state.pc = 1
```

```
    state.op_s.val[0] = lo
    state.op_s.top = 0
  -> State: 1.3 <-
    state.pc = 2
    state.op_s.val[1] = hi
    state.op_s.top = 1
  -> State: 1.4 <-
    state.pc = 3
    state.op_s.val[2] = lo
    state.op_s.top = 2
  -> State: 1.5 <-
    state.pc = 4
    state.memory[2] = lo
    state.op_s.val[2] = null
    state.op_s.top = 1
  -> State: 1.6 <-
    state.pc = 5
    state.memory[1] = hi
    state.op_s.val[1] = null
    state.op_s.top = 0
  -> State: 1.7 <-
    state.halted = TRUE
```

L'ultimo stato completo risulta essere infatti:

```
================ State ================
  0) -------------------------
  global.inst_upper_bound = 5
  global.n_inst = 6
  instructions.global.inst_upper_bound = 5
  instructions.global.n_inst = 6
  instructions.inst[0] = push
  instructions.inst[1] = load
  instructions.inst[2] = push
  instructions.inst[3] = store
  instructions.inst[4] = store
  instructions.inst[5] = halt
  instructions.arg[0] = 0
  instructions.arg[1] = 0
  instructions.arg[2] = 0
  instructions.arg[3] = 2
  instructions.arg[4] = 1
  instructions.arg[5] = 0
  instructions.ipd[0] = 0
  instructions.ipd[1] = 0
  instructions.ipd[2] = 0
  instructions.ipd[3] = 0
  instructions.ipd[4] = 0
  instructions.ipd[5] = 0
  state.global.inst_upper_bound = 5
  state.global.n_inst = 6
  state.pc = 5
  state.memory[0] = hi
```

```
29    state.memory[1] = hi
30    state.memory[2] = lo
31    state.ipd_s.global.inst_upper_bound = 5
32    state.ipd_s.global.n_inst = 6
33    state.ipd_s.val[0] = null
34    state.ipd_s.val[1] = null
35    state.ipd_s.addr[0] = 0
36    state.ipd_s.addr[1] = 0
37    state.ipd_s.top = -1
38    state.op_s.val[0] = lo
39    state.op_s.val[1] = null
40    state.op_s.val[2] = null
41    state.op_s.val[3] = null
42    state.op_s.val[4] = null
43    state.op_s.top = 0
44    state.env = lo
45    state.halted = TRUE
```

4.2.2.1 Esempio Corretto

Se modifichiamo il bytecode in modo che l'ordine di inserimento e prelievo dallo stack tenga conto dei livelli di sicurezza delle variabili in memoria otteniamo il seguente frammento di codice:

```
1 push 0
2 load x
3 push 0
4 store y
5 store z
6 halt 0
```

Analizzandolo con NuSMV possiamo vedere come stavolta la formula risulti soddifatta:

```
- specification AG (state.memory[1] = lo) is true
```

Analiazzando i vari stati in modo interattivo infatti vediamo che i livelli di sicurezza delle variabili vengono rispettati.

```
1  Chosen state is: 0
2  Trace Description: Simulation Trace
3  Trace Type: Simulation
4    -> State: 1.1 <-
5      instructions.inst[0] = push
6      instructions.inst[1] = load
7      instructions.inst[2] = push
8      instructions.inst[3] = store
9      instructions.inst[4] = store
10     instructions.inst[5] = halt
11     instructions.arg[0] = 0
12     instructions.arg[1] = 0
13     instructions.arg[2] = 0
14     instructions.arg[3] = 1
15     instructions.arg[4] = 2
16     instructions.arg[5] = 0
17     instructions.ipd[0] = 0
```

```
18      instructions.ipd[1] = 0
19      instructions.ipd[2] = 0
20      instructions.ipd[3] = 0
21      instructions.ipd[4] = 0
22      instructions.ipd[5] = 0
23      state.pc = 0
24      state.memory[0] = hi
25      state.memory[1] = lo
26      state.memory[2] = hi
27      state.ipd_s.val[0] = null
28      state.ipd_s.val[1] = null
29      state.ipd_s.addr[0] = 0
30      state.ipd_s.addr[1] = 0
31      state.ipd_s.top = -1
32      state.op_s.val[0] = null
33      state.op_s.val[1] = null
34      state.op_s.val[2] = null
35      state.op_s.val[3] = null
36      state.op_s.val[4] = null
37      state.op_s.top = -1
38      state.env = lo
39      state.halted = FALSE
40      global.inst_upper_bound = 5
41      global.n_inst = 6
42      instructions.global.inst_upper_bound = 5
43      instructions.global.n_inst = 6
44      state.global.inst_upper_bound = 5
45      state.global.n_inst = 6
46      state.ipd_s.global.inst_upper_bound = 5
47      state.ipd_s.global.n_inst = 6
48    -> State: 1.2 <-
49      state.pc = 1
50      state.op_s.val[0] = lo
51      state.op_s.top = 0
52    -> State: 1.3 <-
53      state.pc = 2
54      state.op_s.val[1] = hi
55      state.op_s.top = 1
56    -> State: 1.4 <-
57      state.pc = 3
58      state.op_s.val[2] = lo
59      state.op_s.top = 2
60    -> State: 1.5 <-
61      state.pc = 4
62      state.op_s.val[2] = null
63      state.op_s.top = 1
64    -> State: 1.6 <-
65      state.pc = 5
66      state.op_s.val[1] = null
67      state.op_s.top = 0
68    -> State: 1.7 <-
```

```
69      state.halted = TRUE
```

Vediamo che i valori della memoria non vengono modificati rispetto a quelli iniziali. Infatti l'ultimo stato risulta:

```
1   ================= State =================
2   0) -------------------------
3   global.inst_upper_bound = 5
4   global.n_inst = 6
5   instructions.global.inst_upper_bound = 5
6   instructions.global.n_inst = 6
7   instructions.inst[0] = push
8   instructions.inst[1] = load
9   instructions.inst[2] = push
10  instructions.inst[3] = store
11  instructions.inst[4] = store
12  instructions.inst[5] = halt
13  instructions.arg[0] = 0
14  instructions.arg[1] = 0
15  instructions.arg[2] = 0
16  instructions.arg[3] = 1
17  instructions.arg[4] = 2
18  instructions.arg[5] = 0
19  instructions.ipd[0] = 0
20  instructions.ipd[1] = 0
21  instructions.ipd[2] = 0
22  instructions.ipd[3] = 0
23  instructions.ipd[4] = 0
24  instructions.ipd[5] = 0
25  state.global.inst_upper_bound = 5
26  state.global.n_inst = 6
27  state.pc = 5
28  state.memory[0] = hi
29  state.memory[1] = lo
30  state.memory[2] = hi
31  state.ipd_s.global.inst_upper_bound = 5
32  state.ipd_s.global.n_inst = 6
33  state.ipd_s.val[0] = null
34  state.ipd_s.val[1] = null
35  state.ipd_s.addr[0] = 0
36  state.ipd_s.addr[1] = 0
37  state.ipd_s.top = -1
38  state.op_s.val[0] = lo
39  state.op_s.val[1] = null
40  state.op_s.val[2] = null
41  state.op_s.val[3] = null
42  state.op_s.val[4] = null
43  state.op_s.top = 0
44  state.env = lo
45  state.halted = TRUE
```