Homework 2

gianmaria di rito 544013

May 2021

1 Introduzione

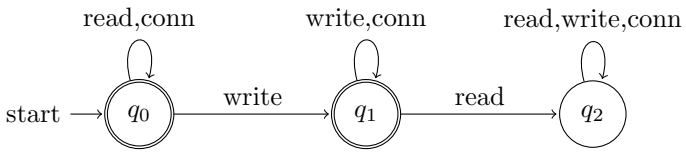
In questo homework bisognava estendere il semplice linguaggio funzionale di base visto a lezione, andando ad includere un meccanismo dinamico che faccia rispettare le politiche di sicurezza locali sulla falsa riga dell' *History-Dependent Access Control*.

2 Scelte progettuali

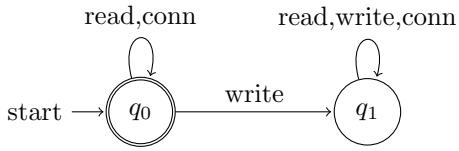
2.1 Local Security Policy

Ho scelta di implementare 3 local security policy; le quali sono definite sull'insieme di azioni rilevanti *read, write, connect*. Esse sono:

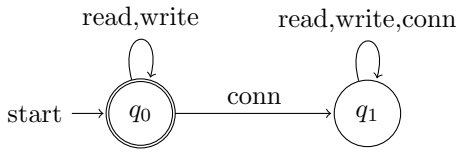
- **don't read after write**: policy che farà si che, se si è all'interno del suo scope, non saranno ammesse operazioni di *read* eseguite in seguito ad operazioni di *write*. Essa nella mia soluzione, è implementata da un automa a stati finiti così formato:



- **don't write**: policy che impedisce un' operazione di scrittura all'interno del suo scope. L'automa a stati finiti che la rappresentà è così fatto:



- **don't connect**: policy che impedisce un'operazione di connect all'interno del suo scope. L'automa a stati finiti che la rappresentà è così definito:



Gli automi a stati finiti sono stati implementati tramite l'aggiunta al linguaggio di base dei tipi:

- **state = int**
- **symbol = char**
- **transition = int * symbol * int**
- **dfa =**

  states : state list;

  sigma : symbol list;

  start : state;

  transitions : transition list;

  accepting : state list;

Essi vengono sfuttati dalla funzione **checkAccepts** che prende in imput una stringa *str* che racchiude la sequenza di azioni rilevanti da controllare
(es. str = wrcc per una sequenza di azioni formata da write, read, connect, connect) e un *dfa* cioè un automa a stati finiti che rappresenta la politica da rispettare. Questa funzione sfrutta altre funzioni ausiliarie come **contains** ed **explode** per decidere se la sequenza di azioni *str* rispetta la politica rappresentata da *dfa*.

2.2 Estensione della Sintassi

La sintassi di base è stata poi arricchita con i costrutti:

- **Letrec of string * expr * expr**
- **Read of string**
- **Write of string * expr**
- **Connect of string**
- **Policy of dfa * expr**

Dove il costrutto *Letrec* è usato per l'implementazione delle funzioni ricorsive.
Read, Write, Connect servono banalmente a rappresentare le azioni di cui riportano il nome.
Policy che prende un *dfa*expr* sarà utilizzato per indicare l'entrata all'interno dello scope della local security policy rappresentata da *dfa*.
Da notare anche la modifica del tipo value che è stato esteso con il costrutto **RecClosure of string * string * expr * value env** il quale andrà a rappresentare una chiusura per le funzioni ricorsive.

2.2.1 Funzioni Ausiliarie

Per l'implementazione dello scope delle policy è stato utilizzato una struttura stack che viene manipolata tramite 3 funzioni ausiliarie:

- **push**: che inserisce un elemento sullo stack
- **top stack**: che restituisce l'elemento in cima allo stack
- **isnotempty**: che controlla se lo stack è non vuoto.

2.3 Estensione dell'interprete "eval"

L'interprete eval è stato esteso innanzitutto, con l'aggiunta di uno stack *st* per l'implementazione dello scope del costrutto sintattico *Policy* tra i suoi dati di input. Esso servirà per tenere traccia dell'entrata nel frame di una politica, soprattutto nel caso di policy innestate una dentro l'altra. Poi è stato modificato il valore restituito dall'interprete *eval* in una coppia *value*string* allo scopo di tenere traccia di tutte le operazioni rilevanti compiute nella storia *eta* dell'esecuzione. Questa storia *eta* è implementata sottoforma di stringa a cui viene concatenato un carattere **w,r,c** a seconda dell'operazione rilevante eseguita. Questo viene fatto all'interno della Write nella Read e nella Connect nel caso in cui il match dell'eval coincida con queste operazioni. Nel caso si debba invece valutare un expr = Policy(dfa,e1) vorra dire che siamo entrati in un frame dove vale una policy *dfa*. Quindi, verra innanzitutto fatto il push sullo stack di questo dfa che rappresenta la policy per tenere traccia del fatto che siamo entrati in questo frame. In seguito verrà valutata l' expr e1.
Sia nel caso di un' expr = Write, Read, Connect che nel caso di un'expr = Policy si valuterà se lo stack non è vuoto. Se questo è vero sapremo che siamo all'intermo dello scope di una policy e quindi possiamo controllare con **checkAccepts** se le azioni rilevanti effettuate fino a questo punto rispettano la policy che si trova in cima allo stack.
L'eval è stato poi esteso all'interno del match con l'expr = Call col caso RecClosure per le chiamate delle funzioni ricorsive e col match con l'expr = LetRec sempre per l'implementazione delle funzioni ricorsive.