Laboratorio 3 - AM21: AR e Stack per la AM di Small21

SOMMARIO: 27 Aprile, 2021

- Materiale: Osservazioni.
- AM21: Il Modello Astratto - Activation Record, AR, e le sue Operazioni.
- Implementazione di AR in Ocaml.
- AM21: Il Modello Astratto - Stack di AR e le sue Operazioni.
- Implementazione dello Stack di AR in Ocaml.
- Attività di Oggi: Esercizi

Materiale: Osservazioni

- small21L3.ml.

  Il Listing distribuito oggi è corregge alcuni punti della soluzione distribuita relativa all'attività del Laboratorio2 del 20/04.

  Le modifiche coinvolgono anche il problema sollevato in uno degli esercizi di ieri ma vanno oltre e forniscono una soluzione che non usa il controllo di eccezioni come era richiesto (esercizio che rimane aperto e da svolgere sulla versione distribuita come soluzione vedi, attività del 26/04, pagina del corso).

Materiale: All'inizio di Ogni Sessione

- Caricare il file Small21LX, distribuito per la sessione, nell'area di sviluppo (terminal, dir. di file system, documents etc).

  Verifica: Caricarlo sul REPL OCaml e Controllare Esecuzione Tests Precedenti.

- Copiarlo in Small21LCopy.

- Seguire la Presentazione delle Attività della Sessione.

- Svogere le Attività modificando il file Small21LX distribuito.

Activation Record, AR.

- Strutturato in 5 componenti:
  $\{\texttt{head}, \texttt{chain}, \texttt{frame}, \texttt{cont}, \texttt{val}\}$:
- **head**, intestazione:
  Contiene `Ide` $+ \{[\texttt{E}], [\texttt{C}]\} +$ [], indicante il tipo di blocco generatore.
- **chain**, static chain:
  Offset espresso da intero `k` $\geq$ `0` (distanza AR primo Frame dei Non Locali)
- **frame**, binding Locali
- **cont**, continuazione:
  Sequenza di Comandi/Statement da eseguire succesivamente
- **val**, return value:
  Contiene `Val` $+$ [], indicante il valore calcolato quanto previsto

AR: Operazioni.

- `Pattern-Matching` su termini AR (i.e. `termine = pattern`)

Rappresentazione.

La rappresentazione stabilisce le strutture di dati con cui sarà rappresentato ogni valore AR.

Useremo tipi algebrici

- **Activation Record: AR**.

```
type head = Name of ide | Exp | Cmd | NoneH;;
type stch = int;;
type frame = env;;
type cnt = cmd list;;
type res = aval option;;

type ar = AR of (head * stch * frame * cnt * res);;

(* Funzione di Astrazione ed Invariante di Rappresentazione per ar:
1) Presentazione (o forma dei valori):
        {t,k,[id1/D1,...,idn/Dn],C,v} per k>=0
2) AF(c) = {AF(t),k,AF(e),AF(C),AF(v)} if c = AR(t,k,e,C,v)
3) I(AR(t,k,e,C,v)) = k>=0
*)
```

Le Operazioni.

In aggiunta all'uso di pattern matching, inseriamo la seguente operazione (vedi Listing small21L3.ml, allegato)

- mkAR5: crea un activation record, dati i corretti parametri.
- head, frame, cnt, res: selezionano i corrispondenti componenti
- toStringAR:

AM21: Modello Astratto - Stack di AR e le sue Operazioni

Stack di AR.

- Struttura di AR ad accesso e selezione LIFO:
  $>$ `ar1`, ..., `ark`]

Operazioni.

- emptyStack: crea stack vuoto
- push: aggiunge un AR
- top: restituisce top, se esiste
- pop: rimuove top, se possibile
- sizeS: calcola numero di AR presenti
- popK: rimuove AR k posizione sotto il top
- bindS: aggiunge binding al frame dello AR top dello stack
- getS: restituisce binding in accordo alla catena statica
- getR: restituisce valore `val` dello AR top dello stack, se possibile
- resetC: reset di componente `cont`, in AR top dello stack, con continuazione fornita come parametro
- resetR: reset di componente `val`, in AR top dello stack, con il valore fornito come parametro
- resetCR: combinazione di resetC e resetR
- addCode: aggiunge il comando, fornito come parametro, in testa al `cont` nello AR top dello stack.

Rappresentazione.

La rappresentazione stabilisce le strutture di dati con cui sarà rappresentato ogni valore AR.

Useremo tipi algebrici

- **Activation Record: AR**.

```
exception EmptyStack;;
exception InvalidUseOfStack of string * stack * int;;
exception InvalidInvocationReturnedValue of ide;;

type stack = Stack of ar list;;

(* ============================================================================ *)
(* Funzione di Astrazione ed Invariante di Rappresentazione per Stack:
1) Presentazione (o forma dei valori): >ar1,...,ark] per k>=0
2) AF(c) = >] if c = Stack []
   AF(c) = >ar1,...,ark]
         iff c = Stack ars and length ars = k and k>0 and
            (hd ars) = ar1 and AF(pop(c)) = >ar2,...,ark]
3) I(c) = true *)
(* ============================================================================ *)
```

Le Operazioni.

In aggiunta alle operazione indicate nel modello (vedi Listing small21L3.ml, allegato)

- toStringStack:
- printStack:

Implementazione Stato di AM21 in OCaaml

- Nei Linguaggi Funzionali come Miranda, Haskell o nel Nucleo Funzionale di Linguaggi Multi-Paradigma come OCaml, i valori denotabili, calcolabili (ovvero, usabili) sono IMMUTABLE ovvero senza Stato.

- Alcuni Sistemi sono descritti utilizzando valori MUTABLE.
  Esempio.
  Activation Record: Modifica di componenti (valore calcolato, valori temporanei, continuation)
  Stack di AR: Aggiunta e Rimozione di un Activation Record.

- Come implementare tali Sistemi con soli valori IMMUTABLE?
  Esempio.
  Rimpiazziamo la modifica di un valore con un nuovo valore che differisce nel componente da modificare:

```
(* Activation Record top di Stack ars *)
      let resetC((Stack ars)as sk) cl =
          match ars with
            | AR(t,k,env,_,retV)::rest -> Stack(AR(t,k,env,cl,retV)::rest)
            | _ -> raise (InvalidUseOfStack("resetC: ", sk, 0))
            ;;
(* Stack di AR *)
      let push (Stack ars) ar =
          Stack(ar::ars)
            ;;
      let pop (Stack ars) =
          match ars with
            | [] -> raise EmptyStack
            | _::rest -> Stack rest
            ;;
```

- Efficienza?
- Sempre Possibile?

Implementazione Stato di AM21 in OCaaml

- Nei Linguaggi Funzionali come Miranda, Haskell o nel Nucleo Funzionale di Linguaggi Multi-Paradigma come OCaml, i valori denotabili, calcolabili (ovvero, usabili) sono IMMUTABLE ovvero senza Stato.

- Alcuni Sistemi sono descritti utilizzando valori MUTABLE.
  Esempio.
  Activation Record: Modifica di componenti (valore calcolato, valori temporanei, continuation)
  Stack di AR: Aggiunta e Rimozione di un Activation Record.

- Come implementare tali Sistemi con soli valori IMMUTABLE?
  Esempio.
  Rimpiazziamo la modifica di un valore con un nuovo valore che differisce nel componente da modificare:

- Efficienza?
  - Structure Sharing: Valori IMMUTABLE possono essere componenti condivisi di tutti i valori strutturati che usano uno stesso valore (vedi il valore calcolato da `push sk ar` e il valore `sk`;
  - I Linguaggi Funzionali usano Structure Sharing come Modello di Memoria.
  - La lista dei Linguaggio Funzionale è LIFO proprio per utilizzare al meglio il Modello di Memoria basato su Structure Sharing.

- Sempre Possibile?
  - NO: Solo quando i valori manipolati dal Sistema non sono Valori "fisici" con Stato.
  - Valori "fisici" con Stato: Apparecchiature con stato interne/esterne controllate dal Sistema.
    Esempio
    Stampante, File System, File, ...
  - In questo caso, questi Valori MUTABLE:
    i) non possono e non devono essere sostituiti;
    ii) forniscono le operazioni per operare con essi
  - I Linguaggi Funzionali usano queste operazioni come operazioni che calcolano il valore unit e richiedono una speciale forma di composizione.