Laboratorio 2 - AM21: Frame, AR, Stack, Memoria, Stato

Sommario: 20 aprile, 2021

- Raccomandazioni e Preliminari sul Materiale Prodotto.
- AM21: Architettura e Ruolo dei Componenti.
- AM21: Il Modello Astratto - Stato.
- AM21: Il Modello Astratto - Operazioni sullo Stato ed i suoi Componenti.
- Implementazione dello Stato.
- Attività di Oggi: Esercizi

SMall21: All'inizio di Ogni Sessione

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

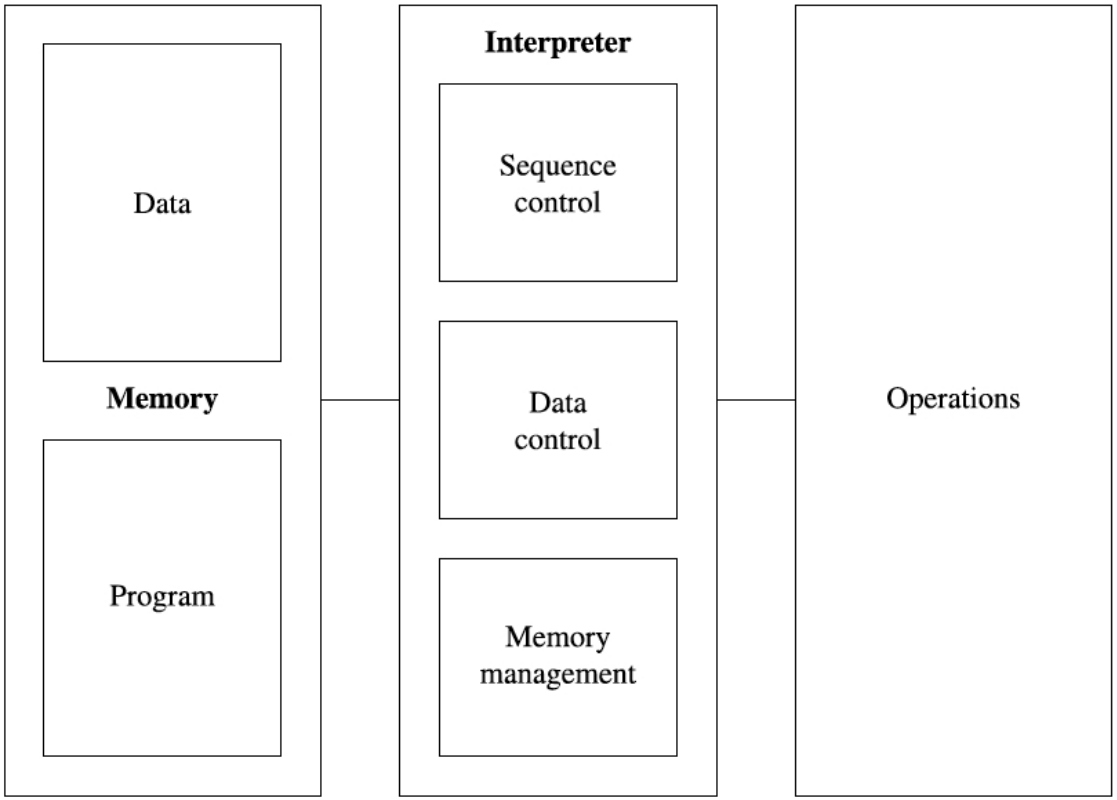
  Verifica: Caricarlo sul REPL OCaml e Controllare Esecuzione Tests Precedenti.

- Copiarlo in Small21LCopy.

- Seguire la Presentazione delle Attività della Sessione.

- Svogere le Attività modificando il file Small21LX distribuito.

AM21: Architettura e Ruolo dei Componenti



- **Program**. Ogni AST di Small21. Esempio: il programma "uno" (laboratorio1.b)
- **Data**. Modello per soli Atomici int, bool (ed eventuali, char, pointer, modificabili e non).
  - Strutturati array (records, list, etc. solo emulati via Data Control)
- **Memory Managment**. Allocazione(/Dealloca) memoria per tutte le richieste.
  - Statica (allocata dinamicamente - dichiarazioni in Seq/Data Control)
  - Stack (Activation Records - Blocks in Seq/Data Control)
  - Heap (solo emulata e solo da estensioni di Small21 - costrutto new e simili)
- **Data Control**. Gestisce Frame e interpreta i binding per accedere ai dati.
- **Sequence Control**. Gestisce Activation Frame e Controlla la AM

AM21: Il Modello Astratto - Lo Stato

- **Frame** per identificatori Locali di Entità Denotabili (costanti, variabili, label, array, (nuovi tipi), procedure/funzioni,...)
  - Indicato da una sequenza finita di **bindings** separati da "," racchiusa in parentesi quadre:
    $$[\mathtt{Ide}_1/\mathtt{Den}_1, ..., \mathtt{Ide}_k/\mathtt{Den}_k]$$

- **Stack di AR** per Blocchi.
  - Ogni Activation Record è indicato da una quintupla di componenti separati da "," racchiusa in parentesi grafe:
    $$\{\mathtt{head}, \mathtt{chain}, \mathtt{frame}, \mathtt{con}, \mathtt{val}\}$$

- **Memoria** Statica per Allocare Variabili ed, al momento, i Componenti modificabili di array dei tipi interi e bool
  - Indicata da sequenza finita di **words** separate da "," racchiusa in parentesi quadre:
    $$[\mathtt{loc}_1 \leftarrow \mathtt{Mv}_1, ..., \mathtt{loc}_k \leftarrow \mathtt{Mv}_k]$$

- **Stato** è rappresentato una coppia (Δ, μ), indicante Stack e Memoria.
  - Lo Stato è indicato con la variabile σ (anche con pedici)

AM21: Il Modello Astratto - Le Operazioni

- **Relazione tra Stato, Semantica ed Operazioni**.
  - Lo Stato esprime tutte e sole le configurazioni della AM21
  - Computazione di un programma Small21 P a partire da uno stato S è la Sequenza di Stati che la Semantica associa a P valutatonello stato iniziale S.
  - La Semantica è espressa nelle sole operazioni di AM21 (che definiremo a partire da oggi)
- **Funzioni Semantiche e Sistema di Inferenza associato**.
  - **dclSem** : $\texttt{Dcl} \rightarrow \texttt{State} \rightarrow \texttt{Type} * \texttt{State}$
    $(\forall \texttt{d}, \sigma)$ **dclSem**$(\texttt{d}, \sigma) = (\texttt{t}, \sigma')$ **iff** $\langle \texttt{d}, \sigma \rangle \rightarrow (\texttt{t}, \sigma') \in \textbf{Sem}_{\textbf{DCL}}$
  - **expSem** : $\texttt{exp} \rightarrow \texttt{State} \rightarrow \texttt{Type} * \texttt{Val} * \texttt{State}$
    $(\forall \texttt{e}, \sigma)$ **expSem**$(\texttt{e}, \sigma) = (\texttt{t}, \texttt{v}, \sigma')$ **iff** $\langle \texttt{e}, \sigma \rangle \rightarrow \lfloor \texttt{t}, \texttt{v}, \sigma' \rfloor \in \textbf{Sem}_{\textbf{EXP}}$
  - **dexpSem** : $\texttt{dexp} \rightarrow \texttt{State} \rightarrow \texttt{Type} * \texttt{DVal} * \texttt{State}$
    $(\forall \texttt{e}_\texttt{d}, \sigma)$ **expSem**$(\texttt{e}_\texttt{d}, \sigma) = (\texttt{t}_\texttt{d}, \texttt{v}_\texttt{d}, \sigma')$ **iff** $\langle \texttt{e}_\texttt{d}, \sigma \rangle \rightarrow \lfloor \texttt{t}_\texttt{d}, \texttt{v}_\texttt{d}, \sigma' \rfloor \in \textbf{Sem}_{\textbf{DEXP}}$
  - **cmdSem** : $\texttt{cmd} \rightarrow \texttt{State} \rightarrow \texttt{Type} * \texttt{State}$
    $(\forall \texttt{c}, \sigma)$ **cmdSem**$(\texttt{c}, \sigma) = (\texttt{t}, \sigma')$ **iff** $\langle \texttt{c}, \sigma \rangle \rightarrow (\texttt{t}, \sigma') \in \textbf{Sem}_{\textbf{CMD}}$
- **Stato: Strutture, Operazioni e Notazione**
  - (Δ, μ) : Stato con Stack di AR, Δ, e Store, μ.
  - Costruttori: Tutte le strutture del Modello (Frame, AR, ...) hanno costruttori che forniscono Termini (Algebra libera) con cui esprimere tutte e sole le strutture di AM21.
  - Operazioni: Tutte le strutture del Modello hanno operazioni di Accesso, Selezione, Modifica componenti per Pattern-Matching.
- **Stack di AR**
  - $>\texttt{ar}_{\texttt{top}} \ldots \texttt{ar}_1]$, struttura LIFO, con $\texttt{ar}_{\texttt{top}}$ ultimo inserito e $\texttt{top} \geq 0$
  - $>]$, Stack vuoto.
  - $\#\Delta$, calcola size dello Stack, ie. numero di ar contenuti in esso.
  - `Pattern-Matching` su termini Stack.

AM21: Il Modello Astratto - Le Operazioni 2

- **Activation Record, AR**.
  - $\{\texttt{head}, \texttt{chain}, \texttt{frame}, \texttt{cont}, \texttt{val}\}$: AR a 5 componenti
  - `Head`, intestazione: `Ide` $+ \{[\texttt{E}], [\texttt{C}]\} + []$
  - `Chain`, static(/dynamic) chain: Offest espresso da intero `k` ≥ 0
  - `Frame`, di ambiente: Vedi sotto
  - `Cont`, continuazione: Sequenza di comandi
  - `Val`, return value: `Val` $+ []$
  - `Pattern-Matching` su termini AR.
- **Frame: Operazioni**.
  - [] Crea un frame vuoto per un AR.
  - $[\texttt{I/D}] \otimes \Delta$ aggiunge il binding `[I/D]` al frame di $\texttt{ar}_{\texttt{top}}$ di Δ
  - $\Delta(\texttt{I})$ denotazione, D, del binding di `I` in Δ, se esiste ed in accordo allo scope di Small21.
  - `Pattern-Matching` su termini frame
- **Store: Operazioni**
  - $\triangleright(\mu, \texttt{n})$ alloca in μ, `n` locazioni libere, in sequenza da `loc`, modifica μ in μ', restituisce $(\texttt{loc}, \mu')$
  - $\triangleleft(\mu, \rho)$ dealloca da μ, le locazioni in ρ, che tornano allocabili. Restituisce la memoria modificata
  - $\mu(\texttt{loc})$ (`loc` deve essere una locazione giá allocata) restituisce il valore di `loc`
  - $\mu[\texttt{loc} \leftarrow \texttt{n}]$ (`loc` deve essere giá allocata) modifica il valore di `loc` con il valore `n`
  - $\texttt{loc} \oplus \texttt{k}$ locazione k words dopo loc.
  - `Pattern-Matching` su termini store.
- **Pattern-Matching**
  - Termine = Pattern: Lega le variabili del Pattern a termini quando può essere reso identico al Termine. Fallisce altrimenti.
  - Uso. Posto in premessa di regole di inferenza, seleziona componenti del termine con cui instanziare la regola. In caso di fallimento rende la regola inapplicabile.

AM21: Il Modello Astratto - Le Operazioni 3

- **Valori Calcolabili e Memorizzabili**.
  - $\texttt{INT} = \{0, 1, -1, \ldots$
  - $\texttt{BOOL} = \{\texttt{Cast(Bool, 0)}, \texttt{Cast(Bool, 1)}\}$
  - $\texttt{DVal} = \{\texttt{loc}_{\texttt{i}}^{\texttt{t}} \mid \texttt{i} \in [0..\texttt{StoreSize}], \texttt{t} \in \{\texttt{Int}, \texttt{Bool}, [\texttt{Arr}]\,\texttt{t}'\,\texttt{k}\}\}$ – memorizzabili per puntatori

- **Denotazioni**.
  - $(\texttt{t}, \texttt{v})$ per costanti: $\texttt{t} \in \texttt{Type}$, $\texttt{v} \in \{\texttt{t}\}$.
  - $([\texttt{Mut}]\,\texttt{t}, \texttt{loc}_{\texttt{t1}})$ per modificabili: $[\texttt{Mut}]\,\texttt{t} \in \texttt{Type}$, $\texttt{loc}_{\texttt{t1}}$ locazione per valori (memorizzabili) in $\{\texttt{t}\}$ Per i castable il valore utilizzato per la rappresentazione: Ad es. $([\texttt{Mut}]\,\texttt{Bool}, \texttt{loc}_{\texttt{Int}})$ per una variabile booleana quando i booleani sono memorizzabili ma non implementati (primitivi).
  - $([\texttt{Mut}]\,([\texttt{Arr}]\,\texttt{t}\,\texttt{N}), \texttt{loc}_{\texttt{t1}})$ per variabili array: $\texttt{loc}_{\texttt{t1}}$ locazione per valori nell'insieme utilizzato per la memorizzazione (vedi sopra, valori memorizzabili).
  - $\star\texttt{I}, \texttt{t}, (\texttt{f}_1, \ldots, \texttt{f}_\texttt{k}), \texttt{d}, \texttt{c}, \texttt{k}\star$ closure per Procedure e Funzioni Procedurali con parametri.
    $\texttt{I}$ = nome, $\texttt{t}$ = tipo, $(\texttt{f}_1, \ldots, \texttt{f}_\texttt{k})$ = formali, $\texttt{d}$ = dichiarazioni, $\texttt{c}$ = comando, $\texttt{k}$ = posizione AR di dichiarazione nello Stack, della Procedura, Funzione Procedurale, dichiarata.

- **Notazione: Simboli**
  - [], Struttura Vuota o Valore di Default.
  - $\perp$, Valore Indefinito.
  - σ (anche con pedici, apici), Stato.
  - Δ (anche con pedici, apici), Stack di AR.
  - μ (anche con pedici, apici), Store.
  - ρ (anche con pedici, apici), Frame.
  - N (anche con pedici, apici), Intero.
  - I (anche con pedici, apici), Identificatore.
  - D, den (anche con pedici, apici), Denotazione.
  - loc, l, loc^{t} (anche con pedici), Locazione di memoria o word (di tipo t).
  - ff, Sequenza, non vuota, parametri formali.
  - aa, Sequenza, non vuota, parametri attuali.

Implementazione: Memoria, Frame e loro operazioni

Dobbiamo estendere AM21 con le strutture del Modello Astratto che definiscono l'Architettura della Macchina per Small21.

Lo facciamo partendo dai componenti base dello stato: Memoria e Frame (o Ambiente dei Locali)

Iniziamo QUI, con il fornire segnature Ocaml alle operazioni previste sui valori Memoria e Frame

Continuiamo DOPO, con il fornire una rappresentazione adeguata per questi valori

- **Memoria**.
  - **allocate**(μ, **n**): alloca `n words` (per interi) in sequenza
  - **upd**(μ, **loc**, **n**): (alias, $\mu[\texttt{loc} \leftarrow \texttt{n}]$) modifica il valore di una locazione
  - **getStore**(μ, **loc**): (alias, $\mu(\texttt{loc})$) fornisce il valore di una locazione
  - **emptyStore**(): crea uno store iniziale con `words` libere, a valore indefinito

- **Frame (o Ambiente Locale)**.
  - **bind**(ρ, **ide**, **den**): (alias, $[\texttt{ide}/\texttt{den}]\circ\rho$) aggiunge un nuovo binding
  - **getEnv**(ρ, **ide**): (alias, $\rho(\texttt{ide})$) valore denotabile di un binding
  - **emptyEnv**(): crea un'ambiente senza bindings

Implementazione: Rappresentazione della Memoria

La rappresentazione stabilisce le strutture di dati con cui sarà rappresentato ogni valore Memoria e ogni valore Frame.

Useremo tipi algebrici

- **Memoria**.

```
type store = Store of int * (loc -> mval);;

(* Funzione di Astrazione ed Invariante di Rappresentazione per store:
1) Presentazione (o forma dei valori): [l1<-Mv1,...,lk<-Mvk] per k>=0
2) AF(c) = [] if c=Store(0,g)
   AF(c) = [l1<-Mv1,...,lk<-Mvk]
               if c=Store(k,g) and (perogni i\in[1..k], li=Loc(i-1) and (g li)=Mvi).
3) I(Store(n,g)) = 0≤n<=storeSize and
        (perogni i\in[1..n], li=Loc(i-1) and (g li) = Mval u and u\in [int]) and
        (perogni i\in[n+1..storeSize], li=Loc(i-1) and (g li) = Undef
*)
```

- **Frame**.

Implementazione: Rappresentazione del Frame

La rappresentazione stabilisce le strutture di dati con cui sarà rappresentato ogni valore Memoria e ogni valore Frame.

Useremo tipi algebrici

- **Memoria**.

- **Frame**.

```
type env = Env of ide list * (ide -> dval);;

(* Funzione di Astrazione ed Invariante di Rappresentazione per env:
   Modificata rispetto a SmallC: Includiamo anche il dominio ide list
   La presentazione non cambia. Cambiamenti nell'invariante e nelle
   operazioni (confronta definizione e loro uso)
1) Presentazione (o forma dei valori): [id1/D1,...,idk/Dk] per k>=0
2) AF(c) = [] if c = Env(l,g) and (perogni id, g(id) = Unbound)
   AF(c) = [id1/D1,...,idk/Dk]
         if c = Env(l,g) and (g(idi) = Di per i\in[1..k],
                              g(x) = Unbound altrimenti).
3) I(c) = (c = Env(l,g) => (List.mem x l) iff (g(x) != Umbound)
*)
```

SMall21: 3 Esercizi

Esercizio (7)

Nello spazio riservato ai "Tests: Semantic Structures", si mostri come dovrebbero essere usate le operazioni sulla memoria per scrivere un'espressione eMem che calcoli:

$[\mathtt{l}_1 \leftarrow \mathtt{m}_1, \mathtt{l}_2 \leftarrow \mathtt{m}_2]$ *dove:* $\mathtt{m}_1$ *sia il memorizzabile 12, ed,* $\mathtt{l}_2$ *sia la prima di 2 locazioni di un array contenente gli interi -18, 11, in tale ordine.*

Nel fare ciò si consideri il comportamento atteso dalle operazioni sulla memoria e non quello effettivo che, come indicato, è errato. Fatto ciò, si mostri:
(a) Quale memoria è calcolata valutando eMem;
(b) Si giustifichi il risultato ottenuto.

Esercizio (8)

Si correggano le operazioni sullo Store, modificando il file Small21L2 caricato in REPL Ocaml. Quindi:
(a) Si mostri la memoria calcolata dalla valutazione di eMem.

Esercizio (9)

Si correggano le definizioni delle operazioni su Env. Quindi:
(a) si fornisca un'espressione eEnv che calcoli:

$$[\mathtt{id}_1/\mathtt{l}_1, \mathtt{id}_2/\mathtt{l}_2]$$

dove: $\mathtt{id}_1$ *e* $\mathtt{id}_2$ *siano gli identificatori* `p1` *e* `mis`*, mentre le denotazioni siano proprio le locazioni introdotte negli esercizi precedenti.*

SMall21: Al termine di Ogni Sessione

- Rimuovere Small21LCopy se sono stati completati tutti gli esercizi altrimenti completarli.

- La prossima sessione:
  - Completeremo lo Stato con AR, Stack e Non Locali
  - Definiremo Semantica SOS delle dichiarazioni