Laboratorio 1
(Progetto da Realizzare e gli Strumenti da Usare)

Sommario: 27 Marzo, 2020

- SmallC: Definizione e Implementazione
- Definizione: Sintassi e Semantica
- Implementazione: Il linguaggio Funzionale Ocaml.
- Ocaml: Guida all'Istallazione
- Ocaml: Introduzione all'uso per il Laboratorio
- Attività di Oggi: Primi Esercizi
  AST: Rappresentazione dei Programmi
  Sintassi Concreta: Presentazione dei Programmi

SmallC: Definizione ed Implementazione

- **Motivazioni e Obiettivi**.
  -
  -
  -
  -
- **Vincoli**.
  -
  -
  -
  -

SmallC: Definizione ed Implementazione

- **Motivazioni e Obiettivi**.
  - Apprendere la definizione di un Linguaggio di Programmazione e la costruzione di un esecutore per esso
  - Scrivere programmi per una soluzione del problema precedente
  - Programmare in un Linguaggio Funzionale
  - Alla fine: Scrivere programmi, non banali, in SmallC ed ottenere esecuzioni corrette, secondo la semantica di SmallC

- **Vincoli**.
  - SmallC deve essere Imperativo (stato) e Prescrittivo (sequenza)
  - Sviluppabile (da noi) in 20 ore (meno 2 ore di oggi)
  - Sviluppo collegiale (no suddivisioni)
  - Istallazione e apprendimento degli strumenti necessari: Ocaml

Definizione: Sintassi - 1

- **Sintassi Astratta**.
  - Il linguaggio lo usiamo solo attraverso la sintassi astratta.
    - Svantaggi:
    - Vantaggi:
- **Elenchiamo strutture e costrutti:**.
    - Minimo(o quasi):
    - Idea:
    - Elenco:
      - **Variabili** (Valori Modificabili per lo Stato)
      - **Valori**:
        Scalari (o Atomici): Interi
        Strutturati:
      - **Tipi**:
      - **Dichiarazioni**:
      - **Espressioni**:
      - **Comandi**:
      - **Programma**:

Definizione: Sintassi - 1

- **Sintassi Astratta**.
  - Il linguaggio lo usiamo solo attraverso la sintassi astratta.
    - Svantaggi: Pesante scrivere programmi in Sintassi Astratta
    - Vantaggi: L'esecutore non ha bisogno (o quasi) di Front-End
- **Elenchiamo strutture e costrutti:**.
    - Minimo(o quasi): indispensabile per un Linguaggio di Programmazione
    - Idea: Pensiamo al C e vediamo cosa dobbiamo inserire e cosa no
    - Elenco:
      - **Variabili**
      - **Valori**:
        Scalari (o Atomici): Interi `e Booleans`
        Strutturati: Array (statici)
      - **Tipi**: `int e bool` (e `Sistema di Tipi`?)
      - **Dichiarazioni**: Variabile, Costante, Array
      - **Espressioni**: Aritmetiche, Relazionali, Logiche, `con Side-Effects` via `Operatore di Assegnamento`
      - **Comandi**: Assegnamento, Condizionale, Sequenza-di-comandi, Iteratore non-determinato (while?, `for`?)
      - **Programma**: Seq. di Dichiarazioni seguita da Seq. di Comandi o `Seq.Dichiarazioni/Comandi arbitraria`?
    - Mancano: Tantissimi costrutti C (primi tra tutti, blocchi-inLine, procedura e ricorsione)

Definizione: Sintassi - 2

- **Elenchiamo strutture e costrutti di SmallC:**
  - **Variabili**
  - **Valori**:
    Scalari (o Atomici): Interi e Booleans
    Strutturati: Array
  - **Tipi**: int e bool (con Sistema di Tipi?)
  - **Dichiarazioni**: Variabile, Costante, Array
  - **Espressioni**: Aritmetiche, Relazionali, Logiche, Assegnamento
  - **Comandi**: Assegnamento, Condizionale, sequenza-di-comandi, Iteratore non-determinato (while o for ?)
  - **Programma**: Seq.Dichiarazioni seguita da Seq. di Comandi o Seq.Dichiarazioni/Comandi arbitrariamente disposti

- **Mancano**: Tantissimi costrutti C (primi tra tutti, ...)

- **Ma sono sufficienti per un Linguaggio di programmazione?**
  $\gg$ SI, perchè:
  dim: Ogni progr. di ... può essere riscritto in un prog. di SmallC calcolante la stessa funzione. Infatti, ...

- **Estendiamo/Riduciamo**: Aggiungeremo/taglieremo costrutti, se il tempo lo/non-lo permetterà

Sintassi Astratta: Una grammatica di alberi AT

```
Type ::= [int] | [bool]
TSys ::= [mut] Type | [arr] Type Num | [void] | [terr]      - ??

Dcl ::= [const] Type Ide Exp | [var] Type Ide Exp          - ??
      | [array] Type Ide Num | Dcl [seqD] Dcl

Exp ::= [val] Ide | Num | Ide [↑] Exp
      | Exp [+] Exp | Exp [−] Exp | Exp [∗] Exp | Exp [div] Exp
      | Exp [==] Exp | Exp [<] Exp | Exp [>] Exp
      | [not] Exp | Exp [or] Exp | Exp [and] Exp
      | Exp [=] Exp

Cmd ::= Ide [=] Exp | Ide Exp [←] Exp | Cmd [seqC] Cmd     - ??
      | [ifte] Exp Cmd Cmd
      | [while] Exp Cmd | [for] Exp Exp Exp Cmd            - ??
```

where:
– ↑ costruttore termine (AT) accesso valore componente array;
– ← costruttore termine modifica valore componente array;

Esercizi.

Esercizio (1)

La sintassi astratta omette la categoria Prog, definente la struttura di un programma, e non prevede i seguenti due costrutti:

- *dichiarazione di identificatore ...;*
- *opposto di ...;*
- *comando if_then_*

(a) Si aggiungano tali costrutti alla Sintassi Astratta e la si completi con la categoria Prog.

(b) Si fornisca una sintassi concreta per il linguaggio SmallC la cui sintassi astratta è stata definita ed estesa come sopra.

Esercizio (2)

Nel file "SmallC.ml" allegato, si estenda la definizione del tipo dcl in Ocaml in modo da fornire la sintassi astratta delle dichiarazioni del Linguaggio SmallC. Allo scopo si riveda quanto fatto, in Ocaml, per la sintassi astratta led Lambda Calcolo.

OCaml: Guida all'istallazione e Uso - 1

La nostra piattaforma (Linux, Mac, Windows, ...) deve avere istallato il pacchetto di strumenti di Ocaml.
L'ultimo aggiornamento di OCaml è Ocaml 4.06.0 (per noi è sufficiente una qualunque versione successiva alla 3.0.0)

- **Download e Istallazione**.
  (a) Connettersi al sito https://www.ocaml.org – Oggi è il sito di riferimento per sviluppatori e utenti.
  (b) accedere a Documentation e selezionare installation instructions
  (c) potete scegliere tra `OPAM` e `Ocaml` – contengono gli stessi strumenti.
  Noi faremo uso dell'interprete durante lo sviluppo e del compilatore alla fine
  Io userò l'interprete `ocaml` e il compilatore `ocamlc` del pacchetto `Ocaml`
  (d) Scaricare e istallare secondo la propria piattaforma.
  Seguire l'istallazione che potrebbe richiedere la configurazione o dare informazioni sul PATH utilizzato.
- **OCaml: Documentation and User's Manual.**
  Accedere alle Pagine del Corso, sezione Materiale in Organizzazione Corso.
- **Uso**.
  (a) apertura di una session:
  - L'interprete è invocato a "linea di comando" (da Teminal in OSX, da Cmd in Windows, ...)
  - che apre una session interattiva con prompt # (vedi session allegata)
  - interpreta (esegue) ogni termine racchiuso tra il prompt e il simbolo ';;' (doppio punto-e-virgola)
  - stampa il valore calcolato (termine irriducibile) nella linea successiva
  - stampa il prompt nella successiva linea e rimane in attesa
  - Ogni termine è interpretato sull'ambiente della sessione definito dalle valutazioni precedenti

  (b) chiusura di una session:
  - usare termine (comando), dove n sia un intero: exit(n);;

OCaml: Guida all'istallazione e Uso - 2

La nostra piattaforma (Linux, Mac, Windows, ...) deve avere istallato il pacchetto di strumenti di Ocaml.
L'ultimo aggiornamento fornisce Ocaml 4.04.0 (per noi è sufficiente una qualunque versione successive alla 3.0.0)

- **Download e Istallazione**.
- **Uso**.
  (a) apertura di una session:
  (b) chiusura di una session
  (c) I termini di una session:
    - Espressioni di Ocaml
    - Comandi per controllo interprete: exit (chiusura), #use(caricamento di file di programma), open ....

  (d) Editing del programma:
    - Il codice di un Programma può coinvolgere molti termini, essere scritto in più linee e conservato in più files.
    - I files di codice ocaml devono essere file testo ed hanno suffisso '.ml'.
    - I file possono essere editati con un qualunque editor per file testo.

  (e) Caricamento di un file in una session:
    - # #use "A.ml";;

  (f) Prova: Accesso Sessione, Scrittura ed Esecuzione di un programma:

```
Last login: Thu Mar 26 08:57:20 on console
[host-131-114-220-141:~ marcob$ ocaml
        OCaml version 4.02.2

# let rec gcd x y =
        if x=y then x
        else if x>y then gcd (x-y) y else gcd x (y-x);;
val gcd : int -> int -> int = <fun>
# gcd 3 17;;
- : int = 1
# gcd 64 6
  ;;
- : int = 2
#
```