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

Sommario: 16 Marzo, 2018

- SmallC: Definizione e Implementazione
- Definizione: Sintassi e Semantica
- Implementazione: Il linguaggio Funzionale Ocaml.
- Ocaml: Guida all'Istallazione
- Ocaml: Introduzione all'uso per il Laboratorio

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)
  - Sviluppabiie (da noi) in 20 ore (meno 2 0re 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** (per lo stato)
      - **Valori**:
        Scalari (o Atomici): Interi
        Struttrati:
      - **Dichiarazioni**:
      - **Espressioni**:
      - **Comandi**:

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** (per lo stato)
      - **Valori**:
        Scalari (o Atomici): Interi
        Struttrati: Array
      - **Dichiarazioni**: Variabile intera, Costante intera, Array
      - **Espressioni**: Aritmetiche, Relazionali, Logiche
      - **Comandi**: Assegnamento, Condizionale, sequenza-di-comandi
        Iteratore non-determinato (while)
    - Mancano: Tantissimi costrutti C (prima tra tutti, procedura e ricorsione)

Definizione: Sintassi - 2

- **Elenchiamo strutture e costrutti di SmallC:**.
  - **Variabili** (per lo stato)
  - **Valori**:
    Scalari (o Atomici): Interi
    Struttrati: Array
  - **Dichiarazioni**: Variabile intera, Costante intera, Array
  - **Espressioni**: Aritmetiche, Relazionali, Logiche
  - **Comandi**: Assegnamento, Condizionale, sequenza-di-comandi
    Iteratore non-determinato (while)

- **Mancano**: Tantissimi costrutti C (prima tra tutti, procedura e ricorsione)

- **Ma sono sufficienti per un Linguaggio di programmazione?**
  $\gg$ SI, perchè:
  dim: Ogni progr. di ... può essere riscritto in un prog. di Small

- **Estendiamo**: Aggiungeremo costrutti, se il tempo lo permetterà

Sintassi Astratta: Una grammatica di alberi AT

$\texttt{Dcl} ::= [\texttt{const} - (\texttt{Ide}, \texttt{Num})]$	– const è l'etichetta della radice, ovvero l'operatore principale che identifica il costrutto per la dichiarazione di una costante
$\mid [\texttt{var} - (\texttt{Ide}, \texttt{Num})]$	– var stessa cosa per una dichiarazione di variabile
$\mid [\texttt{array} - (\texttt{Ide}, \texttt{Num})]$	– array idem
$\mid [; -(\texttt{Dcl}, \texttt{Dcl})]$	– ; idem ma *brutto nome per un operatore*

La notazione usata per esprimere gli alberi astratti nella grammatica per la sintassi astratta è PRECISA ma NOIOSA:

- le parentesi (,) per racchiudere la lista dei sottoalberi
- il separatore '-' e l'obbligo di scrivere l'operatore a sx dei sottoalberi

Nel caso delle espressioni potremmo trovarci a scrivere: [- - (Exp,Exp)]

Rilassiamo la notazione nel seguente modo:

- Parentesi tonde: abolite;
- Solo l'etichetta della radice racchiusa nelle parentesi [,]
- Radice scritta in forma prefissa, infissa, postfissa

Esempio. Scriveremo Exp [-] Exp per l'albero [- - (Exp,Exp)]

Sintassi Astratta: Una grammatica di alberi AT

$$\texttt{Dcl} ::= [\texttt{const}]\ \texttt{Ide}\ \texttt{Num} \mid [\texttt{var}]\ \texttt{Ide}\ \texttt{Num} \mid [\texttt{array}]\ \texttt{Ide}\ \texttt{Num} \mid \texttt{Dcl}\,[\texttt{seqD}]\,\texttt{Dcl}$$

$$\begin{aligned}\texttt{Exp} ::= {} & [\texttt{val}]\ \texttt{Ide} \mid \texttt{Num} \mid \texttt{Ide}\,[\uparrow]\,\texttt{Num} \\ & \mid \texttt{Exp}\,[+]\,\texttt{Exp} \mid \texttt{Exp}\,[-]\,\texttt{Exp} \mid \texttt{Exp}\,[*]\,\texttt{Exp} \mid \texttt{Exp}\,[\texttt{div}]\,\texttt{Exp} \\ & \mid \texttt{Exp}\,[==]\,\texttt{Exp} \mid \texttt{Exp}\,[<]\,\texttt{Exp} \mid \texttt{Exp}\,[>]\,\texttt{Exp} \\ & \mid [\texttt{not}]\,\texttt{Exp} \mid \texttt{Exp}\,[\texttt{or}]\,\texttt{Exp} \mid \texttt{Exp}\,[\texttt{and}]\ \texttt{Exp}\end{aligned}$$

$$\begin{aligned}\texttt{Cmd} ::= {} & \texttt{Ide}\,[=]\,\texttt{Exp} \mid \texttt{Ide}\ \texttt{Exp}\,[\leftarrow]\,\texttt{Exp} \mid \texttt{Cmd}\,[\texttt{seqC}]\,\texttt{Cmd} \\ & \mid [\texttt{ifte}]\ \texttt{Exp}\ \texttt{Cmd}\ \texttt{Cmd} \mid [\texttt{while}]\ \texttt{Exp}\ \texttt{Cmd}\end{aligned}$$

where:
– $\uparrow$ costruttore termine (AT) accesso valore componente array;
– $\leftarrow$ 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:
(a) dichiarazione di identificatore non inizializzato;
*(b) comando if_then_*
Si aggiungano tali costrutti alla Sintassi Astratta e la si completi con la categoria Prog.

Esercizio (2)

Si fornisca in C una rappresentazione degli alberi AT della nostra grammatica:
(a) Limitandoci alle sole dichiarazioni;
(b) Fornendo un main per la creazione dell'albero `[const - (Ide,Num)];[array - (Ide,Num)]` *e la sua stampa.*

Esercizio (3)

Si fornisca una sintassi concreta per il linguaggio SmallC la cui sintassi astratta è stata definita ed estesa come nell'esercizio1, sopra.

Sintassi Astratta di SmallC (Soluzione Esercizio1)

$$\texttt{Dcl} ::= [\texttt{const}]\ \texttt{Ide}\ \texttt{Num} \mid [\texttt{var}]\ \texttt{Ide}\ \texttt{Num} \mid [\texttt{array}]\ \texttt{Ide}\ \texttt{Num} \mid \texttt{Dcl}\ [\texttt{seqD}]\ \texttt{Dcl}$$

$$\begin{aligned}\texttt{Exp} ::= {} & [\texttt{val}]\ \texttt{Ide} \mid \texttt{Num} \mid \texttt{Ide}\ [\uparrow]\ \texttt{Num} \\ & \mid \texttt{Exp}\ [+]\ \texttt{Exp} \mid \texttt{Exp}\ [-]\ \texttt{Exp} \mid \texttt{Exp}\ [*]\ \texttt{Exp} \mid \texttt{Exp}\ [\texttt{div}]\ \texttt{Exp} \\ & \mid \texttt{Exp}\ [==]\ \texttt{Exp} \mid \texttt{Exp}\ [<]\ \texttt{Exp} \mid \texttt{Exp}\ [>]\ \texttt{Exp} \\ & \mid [\texttt{not}]\ \texttt{Exp} \mid \texttt{Exp}\ [\texttt{or}]\ \texttt{Exp} \mid \texttt{Exp}\ [\texttt{and}]\ \texttt{Exp}\end{aligned}$$

$$\begin{aligned}\texttt{Cmd} ::= {} & \texttt{Ide}\ [=]\ \texttt{Exp} \mid \texttt{Ide}\ \texttt{Exp}\ [\leftarrow]\ \texttt{Exp} \mid \texttt{Cmd}\ [\texttt{seqC}]\ \texttt{Cmd} \\ & \mid [\texttt{ifte}]\ \texttt{Exp}\ \texttt{Cmd}\ \texttt{Cmd} \mid [\texttt{if}]\ \texttt{Exp}\ \texttt{Cmd} \mid [\texttt{while}]\ \texttt{Exp}\ \texttt{Cmd}\end{aligned}$$

$$\texttt{Prog} ::= [\texttt{prog}]\ \texttt{Dcl}\ \texttt{Cmd} \mid [\texttt{progN}]\ \texttt{Cmd}$$

where:
– $\uparrow$ costruttore termine (AT) accesso valore componente array;
– $\leftarrow$ costruttore termine modifica valore componente array;

Sintassi Astratta di SmallC (Soluzione Esercizio2)

Abbiamo 2 alternative:

- Partiamo da Zero (ignorando il lavoro fatto sui Parse Tree)
  - Così è fatto nella pratica dei FrontEnd;
  - Ogni categoria grammaticale è trattata come un tipo con: proprie strutture (di albero) e proprie operazioni.

- Partiamo dai Parse Tree e ...
  - Rivediamo la struttura per rappresentare AT: diversamente dai ParseTree gli AT non hanno $\{\texttt{T}, \texttt{NT}\}$;
  - Come un'unico tipo di valore gli AT delle differenti categorie grammaticali.

* Potremmo esercitarci con entrambi gli approcci
* Oggi seguiremo il secondo e lo confronteremo con quello che daremo in OCaml

Sintassi Concreta di SmallC (Soluzione Esercizio3)

$\texttt{Dcl} ::= \texttt{ide} = \texttt{num} \mid \texttt{var ide} = \texttt{num} \mid \texttt{var ide} \mid \texttt{ide[num]}$
$\texttt{Dcls} ::= \epsilon \mid \texttt{Dcl; Dcls}$

$\texttt{ExpB2} ::= \texttt{ExpB2 or ExpB1} \mid \texttt{ExpB1}$
$\texttt{ExpB1} ::= \texttt{ExpB1 and ExpB} \mid \texttt{ExpB}$
$\texttt{ExpB} ::= \texttt{notExpB} \mid \texttt{ExpR}$
$\texttt{ExpR} ::= \texttt{ExpR} == \texttt{ExpA2} \mid \texttt{ExpR} < \texttt{ExpA2} \mid \texttt{ExpR} > \texttt{ExpA2} \mid \texttt{ExpA2}$
$\texttt{ExpA2} ::= \texttt{ExpA2} + \texttt{ExpA1} \mid \texttt{ExpA2} - \texttt{ExpA1} \mid \texttt{ExpA1}$
$\texttt{ExpA1} ::= \texttt{ExpA1} * \texttt{ExpA} \mid \texttt{ExpA1divExpA} \mid \texttt{ExpA}$
$\texttt{ExpA} ::= \texttt{ide} \mid \texttt{num} \mid \texttt{ide[ExpA2]} \mid \texttt{(ExpB2)}$

$\texttt{Cmd} ::= \texttt{if ExpB2 then Cmd else Cmd} \mid \texttt{OtherCmd}$
$\texttt{OtherCmd} ::= \texttt{if ExpB2 then OtherCmd} \mid \texttt{NonCondionalCmd}$
$\texttt{NonCondionalCmd} ::= \texttt{ide} = \texttt{ExpA2} \mid \texttt{Ide[ExpA2]} = \texttt{ExpA2}$
$\mid \texttt{while ExpB2 Cmd} \mid \{\texttt{Cmd; Cmds}\} \mid \{\texttt{Cmd}\}$[1]
$\texttt{Cmds} ::= \texttt{Cmd} \mid \texttt{Cmd; Cmds}$

$\texttt{Prog} ::= \texttt{Dcls} \{\texttt{Cmds}\}$

where:
- `ide`, `num`, `=`, `var`, ... sono tutte categorie lessicali (contenenti 1 solo lessema, ad eccezione di ide e num).

[1] oppure, | `while ExpB2 Cmd` | {`Cmds`}

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
  Io userò l'interprete `ocaml` e il compilatore `ocamlc` del pacchetto `Ocam`
- **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'ultima aggiornamento supporta 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";;