Laboratorio 1: Il Linguaggio Small21

Sommario: 12 e 13 Aprile, 2021

- Small21: Definizione e Implementazione
- Definizione: Sintassi e Semantica
- Attività di Oggi: Primi Esercizi
  - Macchina Astratta e AST:
    Rappresentazione dei Programmi nella AM
  - Sintassi Concreta:
    Presentazione dei Programmi
    Una Grammatica Non Ambigua CF

Small21: Definizione ed Implementazione

- **Motivazioni e Obiettivi**.
  - Apprendere la Definizione di un Linguaggio Imperativo:
    Small21
  - Scrivere Programmi per la Definizione di una AM21:
    Strutture ed Esecutore di Small21
  - Programmare (intero sviluppo) in un Linguaggio Funzionale:
    OCaml - Nucleo Funzionale
  - Scrivere programmi, non banali, in (AST) Small21
    Interfacce[1] ed **Espressività**[2]
  - Ottenere esecuzioni corrette, secondo la Semantica di Small21
    Verifica e Correttezza
  - Estendere e/o Modificare Small21 e/o AM21
    Progetto/Seminario individuale di Fine Corso

- **Vincoli**.
  - Small21 deve essere Imperativo (Valori con Stato) e Prescrittivo (Sequenza)
  - Sviluppabile (da noi) in 20 ore (meno 2 ore di oggi)
  - Sviluppo collegiale (no suddivisioni)

---

[1] La forma con cui si comunica con AM21: ASTs e (richiesta della) loro Computazione.

[2] Gli algoritmi implementabili in Small21 senza ricorrere ad emulazioni di Nuove Strutture di Controllo.

Small21: Criteri Generali della Definizione - 1

- **Sintassi Astratta e Concreta**.
  - Programmi: Scriviamo in S.A. ma Presentiamo in S.C.
    - Svantaggi:
    - Vantaggi:

- **Elenchiamo strutture e costrutti:**.
    - Minimo (non troppo):
    - Idea:
    - Elenco:
      - **Variabili** (Valori Modificabili)
      - **Valori** (o Dati):
        Scalari (o Atomici):
        Strutturati:
      - **Tipi**:
      - **Dichiarazioni**:
      - **Espressioni**:
        Operatori Dati, Controllo Sequenza e Astrazioni
      - **Comandi**:
        Operatori Stato, Controllo Sequenza e Astrazioni
      - **Programma**:

Small21: Struttura e Principali Costrutti

- **Sintassi Astratta e Concreta**.
  - I Programmi li scriviamo solo in Sintassi Astratta.
    - Svantaggi: Pesante scrivere programmi in Sintassi Astratta
    - Vantaggi: AM21 non ha bisogno (di Lexer, e Parser) di Front-End
  - I Programmi li presentiamo (anche) in Sintassi Concreta.

- **Elenchiamo Strutture e Costrutti:**.
  - Minimo(o quasi): Indispensabile per definire le funzioni in $\mathcal{F}$
  - Idea: Partiamo dal nucleo del C - estendiamo e generalizziamo.
  - Elenco:
    - **Tipi**: `int`, `bool`, `lab`, ..., `Sistema di Tipi` (Programmi Ben Tipati)
    - **Variabili** (valori Modificabili o con Stato)
    - **Label** (etichette per Posizioni in sequenza programma)
    - **Valori**:
      Scalari (o Atomici): `Interi`, `Booleans`, ...
      Strutturati: `Array`, ...
    - **Dichiarazioni**: `Variabile`, `Costante`, `Array`, `Procedura-Funzione Ricorsiva`, ...
    - **Espressioni**: Aritmetiche, Relazionali, Logiche, `Invocazione`, Blocco-Exp, ...
    - **Comandi**: `Assegn.`, `Cond.`, `Sequenza`, `goto`, `Blocco-Cmd`, `Invocazione`, Iteratori, ...
    - **Programma**: `Blocco-Cmd`

Sintassi Astratta: Una grammatica di alberi AST

```
Type ::= [int] | [bool] | [lab] | [void]
       | [mut] Type | [arr] Type Num | [terr]
       | [abs] TypeSeq

Dcl ::= [var] Type Ide Exp | [const] Type Ide Exp
       | [array] Type Ide | Dcl [seqD] Dcl
       | [pcd] Type Ide FPars BlockC | [emptyD]

Exp ::= DExp
       | [num] Num | Exp [+] Exp
       | Exp [==] Exp
       | [true] | [false] | Exp [or] Exp
       | Exp [>] Exp | Exp [<] Exp
       | [apply] Ide APars | [emptyE]
DExp ::= [val] Ide | DExp [↑] Exp | Ide [↑1] Exp

Cmd ::= Lab [:] Stm | Stm

Stm ::= DExp [=] Exp | [ifE] Exp Stm Stm | [goto] Lab
       | Stm [seqC] Stm | BlockC | [emptyS]
       | [call] Ide APars | [return] Exp

Prog ::= [prog] Ide Block
```

Sintassi Astratta: Una grammatica di alberi AST - 2

```
TypeSeq ::= Type [::] TypeSeq | Type

Num                          -- token del lessico degli interi
Ide                          -- token del lessico degli identificatori
Lab                          -- token del lessico delle labels di cmd

FPars ::= [fp] PPF Type Ide | [emptyFP]

PPF ::= [value] | [ref]      -- Parameter passing Form

BlockE ::= [blockE] Dcl Exp
BlockC ::= [blockC] Dcl Stm    -- in-line block
Block ::= [block] Dcl CmdSeq
CmdSeq ::= [seqC] Cmd CmdSeq | EC

APars ::= [ap] Exp | [emptyAP]
```

Osservazioni

- [mut]: Costruttore per Tipi di valori mutable. Ad es.: [mut][int], [mut]([arr]([int],30)), [arr]([mut][bool],15).
- ↑: Costruttore di termine di accesso a componente array. Ad es.: `x` ↑ 3 corrisponde, in C, all'espressione `x[3]` quando `x` sia una variabile di tipo [mut]([arr]([mut][int],30)).
- SeqTypes: Il tipo più a sinistra indica il tipo immagine i successivi, quando presenti, i tipi degli argomenti della astrazione funzionale, procedurale o di operatore primitivo.
- FPars e APars: Parametri per astrazioni al momento con al più un solo parametro e solo 2 forme di trasmissione.

Esercizi del 12/04

Esercizio (1)

Nel file "Small21.ml" allegato, si estenda la definizione Ocaml del tipo dcl in modo da fornire la Sintassi Astratta delle dichiarazioni del Linguaggio Small21. Allo scopo si riveda quanto fatto, in Ocaml, per la sintassi astratta del Lambda Calcolo.

Esercizio (2)

Si calcoli in OCaml lo AST di Small21 corrispondente alla seguente sequenza C:

```
const int x = 7;
int z = x + 3;
int A[7][3][5];
int succ(int n){return n+1;}
```

nell'ipotesi che il significato atteso per gli analoghi costrutti sintattici di Small21, sia lo stesso dato in C. In particolare, nel caso di array, si ricordi che il costrutto "[arr] t k" indica una struttura array di k componenti di uno stesso tipo t.
Per le strutture parzialmente definite si usino i costrutti disponibili al momento dello svolgimento.

Esercizi del 12/04 - continua

Esercizio (3)

Si proceda come in [1] con gli AST per exp, cmd, e stm. Quindi si calcoli l'AST di Small21 per la sequenza riportata sotto:

```
loop: if (i==size) goto stop;
sum = sum + A[i];
i = i + 1;
goto loop;
```

sempre nelle ipotesi di esercizio in [2]
Per le strutture parzialmente definite si usino i costrutti disponibili al momento dello svolgimento.

Esercizio (4 - Uno sguardo alla Macchina Astratta AM21)

Nell'implemtazione che abbiamo fin qui dato, com'è strutturata e dov'è la memoria in cui risiedono i programmi di Small21?
a) Quando AM21 è "spenta" (non è in esecuzione): Struttura e Luogo?
b) Quando AM21 é "accesa": Struttura e Luogo?

Esercizi del 13/04

Esercizio (5)

Si fornisca una grammatica CF non-ambigua per la Sintassi Concreta di Small21 di cui abbiamo dato la Sintassi Astratta. La Sintassi Concreta

- *deve rispettare i seguenti vincoli:*
  - *1) associatività a sinistra per operatori aritmetici e logici, binari;*
  - *2) precedenza:* $== \prec$ *or* $\prec$ *and* $\prec$ *not* $\prec$ `<`, `>` $\prec$ `+`, $-_2 \prec$ `*`, $/ \prec -_1$;
  - *3) array monodimensionali ed a componenti dei soli tipi int e bool.*
- *deve avere forma (keyword e notazione) prossima a quella del linguaggio C:*
  Ad esempio, uso di ";"-terminatore in dichiarazioni e comandi.

Esercizio (6)

Nel file "Small21-0b.ml" allegato, si completi la definizione delle funzioni toStringXXX, dove il suffisso XXX indichi il tipo OCaml che implementa l'AST della relativa struttura sintattica di Small21. Allo scopo si riveda quanto fatto, in Ocaml, per la sintassi astratta del Lambda Calcolo e si proceda anche qui in modo incrementale, ovvero: Si applichino le definizione via via completate alle strutture (espressioni, dichiarazioni,...) elencate nella sezione finale "Tests: Abstract Syntax".

Ad esempio, dopo aver caricato il file, e prima di aver apportato modifiche si esegua:

```
toStringDcl 0 d1;;
```

Quindi si modifichi la sola definizione di toStringDcl e si ri-esegua verificando l'aderenza con quanto atteso.

Sintassi Concreta: Una CFG per Small21 (;-terminatore)

$$\texttt{Type} \rightarrow \texttt{Simple} \mid \texttt{void} \mid \texttt{Simple}\ \texttt{[Num]}$$
$$\texttt{Simple} \rightarrow \texttt{int} \mid \texttt{bool}$$

$$\texttt{Dcl} \rightarrow \texttt{const Simple ide} = \texttt{Exp;} \mid \texttt{Simple ide} = \texttt{Exp;}$$
$$\mid \texttt{Type ide;} \mid \texttt{Type ide (Fp) Block}$$
$$\texttt{Dcls} \rightarrow \texttt{Dcl Dcls} \mid \epsilon$$

$$\texttt{Exp} \rightarrow \texttt{ExpB} == \texttt{ExpB} \mid \texttt{ExpB}$$
$$\texttt{ExpB} \rightarrow \texttt{ExpB or ExpB}_1 \mid \texttt{ExpB}_1$$
$$\texttt{ExpB}_1 \rightarrow \texttt{truth} \mid \texttt{ExpR}$$
$$\texttt{ExpR} \rightarrow \texttt{ExpA} > \texttt{ExpA} \mid \texttt{ExpA} < \texttt{ExpA} \mid \texttt{ExpA}$$
$$\texttt{ExpA} \rightarrow \texttt{ExpA} + \texttt{ExpT} \mid \texttt{ExpT}$$
$$\texttt{ExpT} \rightarrow \texttt{num} \mid \texttt{(Exp)} \mid \texttt{ide (Ap)} \mid \texttt{DExp}$$
$$\texttt{DExp} \rightarrow \texttt{ide} \mid \texttt{ide [ExpA]}$$

$$\texttt{Cmd} \rightarrow \texttt{lab : Stm} \mid \texttt{Stm}$$
$$\texttt{Cmds} \rightarrow \texttt{Cmd Cmds} \mid \texttt{Cmd}$$
$$\texttt{Stm} \rightarrow \texttt{if (Exp) Stm} \mid \texttt{OtherStm}$$
$$\texttt{OtherStm} \rightarrow \texttt{if (Exp) OtherStm else Stm} \mid \texttt{NonConditionalStm}$$
$$\texttt{NonConditionalStm} \rightarrow \texttt{DExp} = \texttt{Exp;} \mid \texttt{goto lab;}$$
$$\mid \texttt{return Exp;} \mid \texttt{;} \mid \texttt{ide (Ap);} \mid \texttt{Block}$$
$$\texttt{Stms} \rightarrow \texttt{Stm} \mid \texttt{Stm Stms}$$
$$\texttt{Block} \rightarrow \{\texttt{Dcls Stms}\}$$

Sintassi Concreta: Una CFG per Small21 (;-separatore)

$$
\begin{aligned}
&\texttt{Type} \rightarrow \texttt{Simple} \mid \texttt{void} \mid \texttt{Simple [Num]}\\
&\texttt{Simple} \rightarrow \texttt{int} \mid \texttt{bool}\\
\\
&\texttt{Dcl} \rightarrow \texttt{const Simple ide = Exp} \mid \texttt{Simple ide = Exp}\\
&\qquad\qquad \mid \texttt{Type ide} \mid \texttt{Type ide (Fp) Block}\\
&\texttt{Dcls} \rightarrow \texttt{Dcl; Dcls} \mid \epsilon\\
\\
&\texttt{Exp} \rightarrow \texttt{ExpB == ExpB} \mid \texttt{ExpB}\\
&\texttt{ExpB} \rightarrow \texttt{ExpB or ExpB}_1 \mid \texttt{ExpB}_1\\
&\texttt{ExpB}_1 \rightarrow \texttt{truth} \mid \texttt{ExpR}\\
&\texttt{ExpR} \rightarrow \texttt{ExpA > ExpA} \mid \texttt{ExpA < ExpA} \mid \texttt{ExpA}\\
&\texttt{ExpT} \rightarrow \texttt{num} \mid \texttt{(Exp)} \mid \texttt{ide (Ap)} \mid \texttt{DExp}\\
&\texttt{DExp} \rightarrow \texttt{ide} \mid \texttt{ide [ExpA]}\\
\\
&\texttt{Cmd} \rightarrow \texttt{lab : Stm} \mid \texttt{Stm}\\
&\texttt{Cmds} \rightarrow \texttt{Cmd Cmds} \mid \texttt{Cmd}\\
&\texttt{Stm} \rightarrow \texttt{if (Exp) Stm} \mid \texttt{OtherStm}\\
&\texttt{OtherStm} \rightarrow \texttt{if (Exp) OtherStm else Stm} \mid \texttt{NonConditionalStm}\\
&\texttt{NonConditionalStm} \rightarrow \texttt{DExp = Exp} \mid \texttt{goto lab}\\
&\qquad\qquad \mid \texttt{return Exp} \mid \texttt{;} \mid \texttt{ide (Ap)} \mid \texttt{Block}\\
&\texttt{Stms} \rightarrow \texttt{Stm} \mid \texttt{Stm Stms}\\
&\texttt{Block} \rightarrow \texttt{\{Dcls Stms\}}\\
\\
&\texttt{Prog} \rightarrow \texttt{Program ide \{Dcls Cmds\}}
\end{aligned}
$$