PROGRAMMAZIONE II – A.A. 2013-14 - Progetto Sessione Autunnale - vers. 2

Andrea Corradini & GianLuigi Ferrari

Il progetto ha l'obiettivo di applicare i concetti e le tecniche di programmazione esaminate durante il corso a casi specifici. Il progetto consiste nella progettazione e realizzazione di alcuni moduli software, ed è strutturato in due esercizi di programmazione.

Esercizio 1: Progettazione e sviluppo di un interprete in OCaml

Si consideri un semplice linguaggio funzionale che oltre a operazioni standard su interi e booleani comprende la definizione di un tipo di dati strutturato **record**, con relative operazioni, la cui sintassi concreta è definita dalla seguente grammatica:

(Identificatori)	Ide ::= <definizione standard>	
	IdeList ::= <liste di Ide, sintassi standard di OCaml>	(es: [], [nome;cognome], ….)
(Interi)	Int ::= <definizione standard>	
(booleani)	Bool ::= <definizione standard>	
(Espressioni)	E ::= Ide | Int | Bool	
	| E and E | E or E | not E	(Espressioni Booleane)
	| OP(E,E)	(Espressioni su interi con OP ∈{"+", "-", "*", "=", "<="})
	| Ide -> E	(Funzione con un parametro, non ricorsiva)
	| if E then E else E	(Condizionale)
	| let Ide = E in E	(Blocco let)
	| E(E)	(Applicazione Funzionale)
	| record{ Fields }	(Espressione record)
	| E.Ide	(Selezione)
	| E ~ E	(Similitudine di record)
	| E@IdeList(E)	(Applica una funzione a specifici campi di un record)
	Fields ::= Ide:E | Ide:E; Fields	(Campi di espressione record)

1. Definire una sintassi astratta per il linguaggio, introducendo opportuni tipi di dati OCaml. Si noti che i record possono essere annidati.
2. Definire in OCaml un interprete del linguaggio che rispetti le seguenti specifiche:
    a. L'applicazione funzionale deve essere valutata con la regola di scoping statico.
    b. La valutazione di un'espressione record restituisce un record in cui tutti i campi sono valutati, in un ambiente che comprende anche le associazioni determinate dai campi precedenti. Per esempio, la valutazione di **record{base:5; alt:base*2}** deve restituire **record{base:5; alt:10}**.
    c. L'operazione di selezione **exp.id** restituisce il valore del campo di nome **id** del record **exp**. Non è definita se **exp** non è un record oppure se non ha un campo di nome **id**.
    d. L'espressione **exp1 ~ exp2** è definita solo se entrambi gli argomenti sono record, e vale true se **exp1** e **exp2** hanno campi con gli stessi nomi e nello stesso ordine.

e. L'applicazione **exp1@lst(exp2)** è definita solo se **exp1** è una funzione ed **exp2** è un record. In questo caso il risultato è il record ottenuto applicando la funzione **exp1** a tutti i campi di **exp2** il cui nome è nella lista **lst**.
f. Tutte le altre operazioni hanno il significato visto a lezione. Per quanto non specificato documentare e motivare le scelte fatte nella relazione.

3. Verificare la correttezza dell'interprete progettando ed eseguendo una quantità di casi di test sufficiente a testare tutti gli operatori.
4. Tradurre nella sintassi astratta proposta la seguente espressione, e valutarla con l'interprete.

```
let base  =  2 in
let raddoppia = (x -> base * x) in
let rec1 = record{base:5; alt:base*3}  in
let rec2 = record{base:3; alt:raddoppia(base)} in
let rec3 = raddoppia@[base;alt](rec1)  in
  if rec1 ~ rec2
      then record{a1:rec1.alt;a2:rec2.alt;a3:rec3.alt}
     else rec3
```

Il valore atteso è `record{a1:15;a2:6;a3:30}`

Esercizio 2: Progettazione e realizzazione di un modulo Java

Scrivere un programma che simuli lo scambio di SMS su una rete di telefonia mobile. Il sistema deve consentire di gestire più compagnie telefoniche, ognuna delle quali ha un nome (unico nel sistema) e una tariffa per l'invio di un SMS. Inoltre il sistema deve gestire più cellulari, ognuno dei quali è registrato presso una unica compagnia telefonica, ed ha un proprietario, un numero di telefono, una coda di SMS, un PIN e un credito residuo.

Il programma deve essere costituito da un numero adeguato di classi per modellare le varie entità coinvolte: in particolare ci dovrà essere una classe chiamata Cellulare con l'ovvio significato. Le strutture dati e le funzionalità devono essere distribuite tra le varie classi in modo da garantire l'incapsulamento.

Il sistema deve realizzare funzionalità per due tipologie di utenti: l'operatore e i clienti. L'operatore può creare una nuova compagnia, eliminare una compagnia (e di conseguenza tutti i suoi cellulari), creare un nuovo cellulare, eliminare un cellulare, visualizzare la lista delle compagnie o dei cellulari con le informazioni rilevanti. I clienti (che sono i proprietari dei cellulari) possono leggere i propri SMS o inviare un SMS ad un altro cellulare (identificandosi con il PIN), visualizzare il proprio saldo e ricaricare il credito prepagato. Le varie operazioni possono lanciare opportune eccezioni, i cui nomi sono indicati nell'interfaccia del punto seguente.

1. Per testare le funzionalità realizzate nel progetto (compreso le eccezioni che esse possono lanciare), si realizzi una classe **SMSIntImpl** che implementi l'interfaccia **SMSInt** riportata di seguito.

```
public interface SMSInt {

      // Aggiunge la compagnia telefonica indicata, con la
      // corrispondente tariffa.
      // Restituisce true se l'inserimento ha successo,
      // false se fallisce perche' c'e' gia' una compagnia
      // con quel nome.
public boolean aggiungiCompagnia(String nomeCompagnia, int tariffaSMS);

      // Elimina la compagnia telefonica indicata, con tutti i cellulari
      // ad essa registrati
public void cancellaCompagnia(String nomeCompagnia);
```

```
      // Aggiunge al sistema il cellulare con i dati indicati.
      // Restituisce true se l'inserimento ha successo,
      // false se fallisce perche' c'e' gia' un cellulare
      // con quel numero.
public boolean aggiungiCellulare(String proprietario, String numero,
            String nomeCompagnia, String PIN, int caricoPrepagato);

      // Invia un SMS dal cellulare con numero 'numMittente' a quello con
      // numero 'numDestinatario', con testo 'testo'. Restituisce
      // il credito residuo del mittente, in centesimi di Euro.
      // Lancia un'eccezione InsuffCreditException se il mittente
      // non ha un credito sufficiente, o WrongIdException
      // se 'numMittente' o 'numDestinatario' non sono numeri di
      // cellulari registrati o se il PIN non e' corretto
public int sendSMS(String numMittente, String numDestinatario, String testo,
            String PIN) throws InsuffCreditException, WrongIdException;

      // Restituisce il testo del prossimo SMS del cellulare con
      // numero 'num', cancellandolo dalla coda.
      // Lancia un'eccezione EmptySMSQueueException se non ci sono
      // SMS da leggere, o WrongIdException se 'num' non e' il numero di
      // un cellulare registrato o il PIN e' incorretto
public String readSMS(String num, String PIN) throws EmptySMSQueueException,
            WrongIdException;

      // Restituisce il credito residuo del cellulare individuato da num e PIN,
      // in centesimi di Euro.
      // Lancia l'eccezione se num o PIN non sono validi
public int balance (String num, String PIN) throws WrongIdException;

      // Ricarica il cellulare individuato da num e PIN. Lancia l'eccezione
      // se num o PIN non sono validi
public void reCharge (String num, String PIN, int ricarica)
            throws WrongIdException;
}
```

2. Fornire per la classe Cellulare (che deve contenere almeno le strutture dati per memorizzare gli SMS, la compagnia e il credito residuo) una specifica completa, comprendente l'overview, l'invariante di rappresentazione, e per ogni metodo la dimostrazione che l'invariante è preservato.
3. Realizzare una batteria di test per verificare la correttezza del progetto realizzato.

Modalità di consegna Progetto Sessione Autunnale

- Il progetto deve essere svolto e discusso col docente individualmente. Il confronto con colleghi mirante a valutare soluzioni alternative durante la fase di progetto è incoraggiato,
- Per poter sostenere l'orale di un appello, il progetto deve essere consegnato entro le ore 24 di Venerdì 5 settembre. Il progetto vale solo per l'appello di settembre.
- Il progetto deve essere costituito da:
    - I file sorgente contenenti il codice sviluppato e le corrispondenti batterie di test; tutto il codice deve essere adeguatamente commentato nei file sorgente.
    - Una relazione di massimo una pagina per esercizio, che descrive le principali scelte progettuali ed eventuali istruzioni per eseguire il codice.
- La consegna va fatta inviando per email tutti i file in un archivio. Per il corso A, inviare l'email al Prof. Ferrari con oggetto "[PR2A] Consegna progetto". Per il corso B, inviare l'email al Prof. Corradini con oggetto contenente la stringa "[PR2B] Consegna progetto".

Altre informazioni

- Per quanto riguarda il progetto, i docenti risponderanno solo a eventuali domande riguardanti l'interpretazione del testo, e non valuteranno o commenteranno soluzioni parziali prima della consegna.
- Per eseguire nell'interprete OCaml il codice contenuto nel file **prova.ml**, si può usare il comando dell'interprete **#use "prova.ml"** (il carattere '#' è necessario).