Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

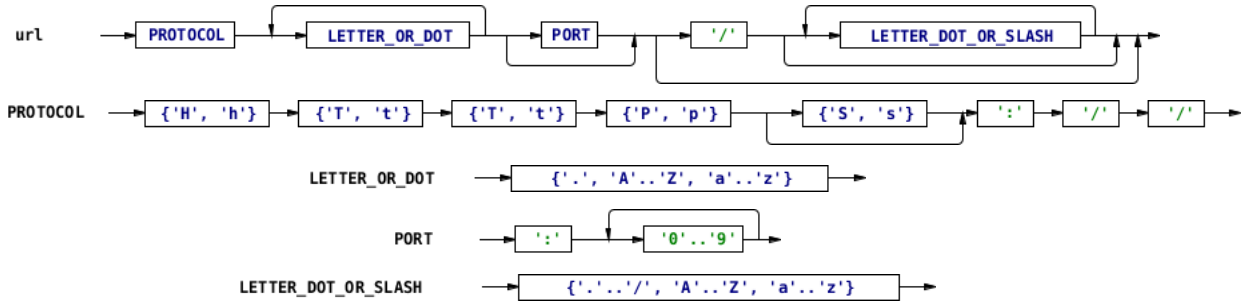
a.a. 2013/2014

6 febbraio 2014

1. (a) Definire la costante `WEB_URL_RE` in modo che il metodo `usedPort` (vedere il listato seguente) possa restituire il numero di porta usato da un URL HTTP(S), passato come stringa. Un URL HTTP(S)[1] deve essere formato come segue:

   - iniziare con le stringhe `"http://"` o `"https://"`, dove qualsiasi lettera può essere minuscola o maiuscola
   - continuare con il nome di un dominio, che per noi è una sequenza arbitraria, ma non vuota, di lettere (minuscole o maiuscole) e punti
   - opzionalmente, continuare con la specifica di un numero di porta, dato dal carattere due punti, una sequenza non vuota di cifre; per esempio, `":42"`
   - continuare con una sequenza arbitraria, anche vuota, di lettere (minuscole o maiuscole), punti (`"."`) e slash (`"/"`). Se questa sequenza è presente (cioé non è la stringa vuota), deve iniziare necessariamente con uno slash.

   Graficamente,



   Alcuni esempi di input/output sono dati nel `main`, dove i commenti indicano l'output delle singole chiamate.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Urls {
    private static final String WEB_URL_RE = ...COMPLETARE...;
    static private Pattern webUrlPattern = Pattern.compile(WEB_URL_RE);

    public static int usedPort(String url) {
        if (url == null)
            throw new IllegalArgumentException("url cannot be null");
        Matcher m = webUrlPattern.matcher(url);
        if (!m.matches())
            throw new IllegalArgumentException("Malformed HTTP(S) URL");
        String specifiedPort = m.group("PORT");
        if (specifiedPort != null)
            return Integer.valueOf(specifiedPort);
        return m.group("SSL") != null ? 443 : 80;
    }

    public static void main(String[] args) {
        System.out.println(usedPort("http://www.google.it")); // 80
        System.out.println(usedPort("https://www.GOOGLE.it/pluto")); // 443
        System.out.println(usedPort("hTTp://www.dibris.unige.it:1234/qui/quo/qua.html")); // 1234
        System.out.println(usedPort("HttPs://www.dibris.unige.it:567/clarabella.php")); // 567
    }
}
```

[1] La specifica vale solo ai fini di questo esercizio, in realtà è molto più complessa.

(b) Mostrare che la seguente grammatica è ambigua.

```
Params  ::=  Empty | NonEmpty
Empty ::= ϵ
NonEmpty ::= Type Id | NonEmpty , NonEmpty
Type := int | char | boolean
Id ::= x | y | z
```

(c) Modificare la grammatica definita al punto precedente in modo che **non sia ambigua** e che il linguaggio generato a partire dal non terminale `Params` **resti invariato**.

2. Considerare la funzione `explode: ('a * 'b) list -> 'a list * 'b list` che, presa una lista di coppie $[(e'_1, e''_1); \ldots; (e'_n, e''_n)]$ restituisce la coppia di liste $([e'_1; \ldots; e'_n], [e''_1; \ldots; e''_n])$.

   Esempio:

```
# explode [(1,2);(3,4);(5,6)]
- : int list * int list = ([1; 3; 5], [2; 4; 6])
# explode [("ciao",true);("mondo",false)]
- : string list * bool list = (["ciao"; "mondo"], [true; false])
```

(a) Definire la funzione `explode` direttamente, senza uso di parametri di accumulazione.

(b) Definire la funzione `explode_accum` direttamente, usando un parametro di accumulazione affinché la ricorsione sia di coda.

(c) Definire la funzione `explode_itlist` come specializzazione della funzione `it_list` così definita:

```
let rec it_list f a = function x::l -> it_list f (f a x) l | _ -> a;;
val it_list : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
```

Se volete, potete utilizzare la funzione standard `List.rev`, che "rovescia" una lista; per esempio:

```
# List.rev [1;2;3]
- : int list = [3; 2; 1]
```

3. Considerare le dichiarazioni dei seguenti tipi, che modellano AST per le espressioni regolari costruite a partire dai seguenti operatori: stella di Kleene `*` (`StarExp`), operatore `?` di opzionalità (`OptionalityExp`), operatore `+` di ripetizione non vuota (`PlusExp`), operatore di unione `|` (`UnionExp`), operatore di concatenazione (`CatExp`), singolo carattere (`SymbolExp`) e stringa vuota (`EmptyStringExp`).

```
public interface Visitor<T> {
        T visit(StarExp e);
        T visit(OptionalityExp e);
        T visit(PlusExp e);
        T visit(UnionExp e);
        T visit(CatExp e);
        T visit(SymbolExp e);
        T visit(EmptyStringExp e);
}
public interface Exp {
        <T> T accept(Visitor<T> v);
        List<Exp> getChildren();
}
public abstract class AbstractExp implements Exp {
        private final List<Exp> children;
        protected AbstractExp(Exp... children) {
                if (children == null)
                        throw new IllegalArgumentException();
                this.children = Collections.unmodifiableList(Arrays.asList(children));
        }
        public List<Exp> getChildren() {
                return children;
        }
}
public class PlusExp extends AbstractExp { // plus operator: exp +
        // DA COMPLETARE (1)
}
public class SymbolExp extends AbstractExp { // single string containing just one character
        private final char symbol;
        public SymbolExp(char symbol) {
                this.symbol = symbol;
        }
        @Override
        public <T> T accept(Visitor<T> v) {
                // DA COMPLETARE (2)
        }
        public char getSymbol() {
                return symbol;
        }
}
```

```
public class OptionalityExp extends AbstractExp { // optionality operator: exp ?
        public OptionalityExp(Exp exp) {
                super(exp);
        }
        @Override
        public <T> T accept(Visitor<T> v) {
                // DA COMPLETARE (3)
        }
}
...
```

(a) Completare le definizioni delle classi `PlusExp`, `SymbolExp` e `OptionalityExp`.

(b) Completare la definizione della seguente classe `MatchEmptyString` che implementa visitor per gli oggetti di tipo `Exp`; la visita di un AST deve restituire il valore **`true`** se e solo ha successo il match tra la stringa vuota e l'espressione regolare rappresentata dall'AST.

```
public class MatchEmptyString implements Visitor<Boolean> {
// DA COMPLETARE (4)
}
```

(c) Completare la definizione della seguente classe `ElimOptPlus` che implementa visitor per gli oggetti di tipo `Exp`; la visita di un AST che rappresenta l'espressione e deve restituire un nuovo AST corrispondente a un'espressione equivalente a e, dove tutte le occorrenze degli operatori `?` e `+` sono state opportunamente sostituite usando gli operatori ϵ, $\star$, `|` e di concatenazione.

Esempio: la visita dell'AST corrispondente a `(a|b)+c?` restituisce l'AST dell'espressione `(a|b)(a|b)*(c|)`.

```
public class ElimOptPlus implements Visitor<Exp> {
        @Override
        public Exp visit(StarExp exp) {
                // DA COMPLETARE (5)
        }
        @Override
        public Exp visit(OptionalityExp exp) {
                // DA COMPLETARE (6)
        }
        @Override
        public Exp visit(PlusExp exp) {
                // DA COMPLETARE (7)
        }
        @Override
        public Exp visit(SymbolExp exp) {
                // DA COMPLETARE (8)
        }
        ...
}
```

4. Considerare le seguenti dichiarazioni di classi Java (che utilizzano le classi dichiarate nell'esercizio precedente):

```
public class P {
        Object m(Exp... exps) {
                return "P.m(Exp...)";
        }
        Object m(Exp e) {
                return "P.m(Exp)";
        }
        Object m(Object e) {
                return "P.m(Object)";
        }
}
public class H extends P {
        String m(Exp e) {
                return ((String) super.m(e)) + " H.m(Exp)";
        }
        String m(AbstractExp e) {
                return ((String) super.m(e)) + " H.m(AbstractExp)";
        }
        <T> String m(Exp e, Visitor<T> v) {
                return ((String) super.m(e)) + "<T> H.m(Exp,Visitor<T>)";
        }
}

import static java.lang.System.out;
public class Test {
        public static void main(String[] args) {
                P p1 = new P();
                H h = new H();
                P p2 = h;
                EmptyStringExp e = new EmptyStringExp();
                Exp e2 = e;
                out.println(...);
        }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `p1.m(null)`

(b) `(String) p1.m()`

(c) `h.m(e)`

(d) `h.m(e,new MatchEmptyString())`

(e) `p2.m(e)`

(f) `p2.m(e2)`