Linguaggi di Programmazione
Modulo di Laboratorio di Linguaggi di Programmazione
Progetto Prolog Novembre 2011

Marco Antoniotti e Giuseppe Vizzari

Prolog: compilazione d'espressioni regolari in automi non deterministici.

Le espressioni regolari – *regular expressions*, o, abbreviando *regexps* – sono tra gli strumenti più utilizzati in Informatica. Un'espressione regolare rappresenta un linguaggio (regolare), ovvero rappresenta in maniera finita un insieme potenzialmente infinito di "stringhe".

Rappresentare le espressioni regolari più semplici in Prolog è molto facile. Le regexp più semplici rappresentano *sequenze* di "simboli" e/o regexps, *alternative* tra simboli e/o regexps, e la *ripetizione* di simboli e/o regexps (quest'ultima è anche detta "chiusura" di Kleene). Se `<re>`, $\texttt{<re}_1\texttt{>}$ e $\texttt{<re}_2\texttt{>}$ sono regexp, in Perl (e prima di Perl in '`ed`' UNIX) allora $\texttt{<re}_1\texttt{><re}_2\texttt{>}$, $\texttt{<re}_1\texttt{> | <re}_2\texttt{>}$ e `<re>*` sono esse stesse regexps. In Prolog, senza disturbare il parser intrinseco del sistema, possiamo rappresentare le regexps semplici così:

- $\texttt{<re}_1\texttt{><re}_2\texttt{>}\ldots\texttt{<re}_k\texttt{>}$ diventa $\texttt{seq(<re}_1\texttt{>, <re}_2\texttt{>, }\ldots\texttt{, <re}_k\texttt{>)}$
- $\texttt{<re}_1\texttt{> | <re}_2\texttt{>}$ diventa $\texttt{or(<re}_1\texttt{>, <re}_2\texttt{>)}$
- `<re>*` diventa `star(<re>)`
- `¬<re>` diventa `bar(<re>)`

Altre regexps utili sono:

- $\texttt{oneof(<re}_1\texttt{>, <re}_2\texttt{>, }\ldots\texttt{<re}_k\texttt{>)}$
  ovvero: *una* delle $\texttt{<re}_i\texttt{>}$
- `plus(<re>)`
  ovvero: *almeno una ripetizione* dell'espressione.

Com'è noto, a ogni regexp corrisponde un automa a stati finiti (non-deterministico o NFA).

A lezione sono anche stati mostrati degli esempi su come rappresentare gli NFA in una base dati Prolog e su come scrivere un predicato che "riconosca" una sequenza di simboli come appartenente al linguaggio riconosciuto (o generato) da un automa.

Richiesta

Lo scopo di questo progetto è di realizzare un compilatore da regexps ad NFA con altre operazioni.

Il predicato principale da implementare è **`nfa_re_compile/2`**. Il secondo predicato da realizzare è **`nfa_recognize/2`**. Infine (o meglio all'inizio) va realizzato il predicato **`is_regular_expression/1`**.

1. **`is_regular_expression(RE)`** è vero quando `RE` è un'espressione regolare. Numeri e atomi (in genere anche ciò che soddisfa **`atomic/1)`**, sono le espressioni regolari più semplici.
2. **`nfa_re_compile(FA_Id, RE)`** è vero quando, dato un identificatore `FA_Id` per l'automa (ovvero un termine Prolog senza variabili), `RE` è compilabile in un automa nella base dati del Prolog.
3. **`nfa_recognize(FA_Id, Input)`** è vero quando l'input per l'automa identificato da `FA_Id` viene consumato completamente e l'automa si trova in uno stato finale. `Input` è una lista di "simboli" dell'alfabeto riconosciuto dall'automa.

Notate che non è necessario specificare l'alfabeto Σ. Per la negazione dovete semplicemente assumere che l'automa non riconosca ciò che è specificato nell'espressione regolare.

Esempi

```
?- nfa_re_compile(42, star(oneof(a, s, d, q))). % Complicato.
true ;
false.


?- nfa_recognize(42, [s, a, s, s, d]).
true ;
false.


?- nfa_re_compile(automa_seq, seq(a, s, d)). % Semplice.
true ;
false.


?- nfa_recognize(automa_seq, [a, s, d]).
true ;
true ;
false.


?- nfa_recognize(automa_seq, [a, s, w]).
false.


?- nfa_recognize(automa_seq, [a, w, d]).
false.


?- nfa_re_compile(12, seq(qwe, rty, uio)). %% Cos'è un "simbolo"?
true ;
false.
```

```
?- nfa_recognize(12, [qwe, rty, uio]).
true ;
false.


?- nfa_recognize(12, [qwe, foo, uio]).
false.


?- nfa_recognize(12, [qwe, rty, a]).
false.


?- nfa_re_compile(nonfa, plus(bar(oneof(qwe, rty, uio)))).
true ;
false.


?- nfa_recognize(nonfa, [a, s, a]).
true.


?- nfa_recognize(nonfa, [a, rty, a]).
false.
```

Suggerimenti

I predicati **`nfa_delta/4`**, **`nfa_initial/2`** e **`nfa_final/2`** sono definiti con un **`FA_Id`** come primo argomento.

Attenzione a com'è gestito il cambio di stato alla presenza del simbolo "vuoto" **`epsilon`**.

Il predicato **`nfa_re_compile`** e i suoi predicati ancillari usano – ça va sans dir – il predicato **`assert/1`** o sue varianti.

Si suggerisce anche di definire dei predicati **`nfa_clear/0`**, **`nfa_clear_nfa/1`**, **`nfa_list/0`** e **`nfa_list/1`** che "puliscano" la base dati e che "listino" la struttura di un automa.

Potrebbe essere utile poter generare identificatori univoci per gli stati dei vari automi. A tal proposito, si suggerisce l'utilizzo del predicato **`gensym/2`** che permette di costruire nuovi atomi caratterizzata da una prima parte costante seguita da un numero auto-incrementante secondo il seguente esempio:

```
?- gensym(foo, X).
X = foo1
?- gensym(foo, Y).
Y = foo2
```

Da consegnare

Dovrete consegnare un file `.zip` (i files `.tar` o `.rar` **non sono accettabili**!!!) dal nome

```
Cognome_Nome_Matricola_LPL_NFA_201111.zip
```

`Nome` e `Cognome` devono avere solo la prima lettera maiuscola, `Matricola` deve avere lo zero iniziale se presente (se avete spazi nel nome o nel cognome, sostituiteli con '_').

Questo file deve contenere una sola directory con lo stesso nome (senza lo `.zip` finale, ovviamente). Al suo interno si devono trovare un file Prolog chiamato '`re-nfa.pl`' ed un file di testo chiamato `README.txt`. In altre parole, questa è la struttura della directory (folder, cartella) una volta spacchettata.

```
Cognome_Nome_Matricola_LPL_NFA_201111
      re-nfa.pl
      README.txt
```

Potete aggiungere altri files, ma il loro caricamento dovrà essere fatto automaticamente al momento del caricamento ("loading") dei files sopracitati.

Le prime righe del file '`re-nfa.pl`' dovranno contenere i nomi e le matricole delle persone che hanno svolto il progetto in gruppo. I gruppi possono essere al massimo di tre persone.

Il termine ultimo della consegna è mercoledì 30 novembre 2011 alle ore 23:59 Zulu Time Zone.

Valutazione

Il vostro programma sarà controllato con un insieme d'espressioni assolutamente arbitrario.

Se i files sorgente non potranno essere letti/caricati nell'ambiente ambiente Prolog, il progetto non sarà ritenuto sufficiente **per tutti i componenti del gruppo!**

Il mancato rispetto dei nomi indicati per i predicati, o anche delle strutture proposte e della semantica esemplificata nel testo del progetto, oltre a comportare ritardi e possibili fraintendimenti nella correzione, può comportare un decremento nel voto ottenuto od un giudizio insufficiente.

Riferimenti

[HMU06] J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd Edition, Addison Wesley, 2006

[Sip05] M. Sipser, *Introduction to the Theory of Computation*, 2nd Edition, Course Technology, 2005