Adaptive DFA – the development of adaptable methods

Dan Popa

Univ. "Vasile Alecsandri", Bacau
Overview

➲ 5 years ago, in [Pop 04] and [Pop 05] the Adaptive DFA was proposed
➲ Various implementations was during this years: using Oberon-2, using C and C++, by the original author and his students
➲ Now we are coming back with a mathematical point of view concerning Adaptive DFA suggested by the use of the VHLL Haskell.
Preliminaries

➲ The mathematics of Adaptive DFA is here described using a notation high related to Haskell.
➲ Functions will have long names: \( f(x) \) will be used together with, for ex: \( \text{funct}(x) \) and even \( \text{funct } x \)
➲ Multiple parameters functions will be written not as \( \text{functia } (a, b) \) but as \( \text{functia } a \ b \)
The sets will be in fact ordered sets with eventually duplicated elements (lists). Ex:

\[ x = [1,4,5] \]
\[ [ x | x <- a | x >3 ] \]
Every program is having some auxiliary functions. Here, they are:

--- Intersection of two lists, reloaded ---------

\[
\text{intersect } a \ b = \ [ \ c1 \mid c1 \leftarrow a, c2 \leftarrow b, c1 == c2 \]
\]
Preliminaries (IV)- The “cradle”

-- Adding spaces at the end of the string s
addspace s = ' ':s++" "

-- Also can be written as (not so fast):
-- addspace s = " "++s++" "


Classes of characters

- According to the paper [Pop 05] where Adaptive DFA was mathematically presented for the first time, the characters processed by an Adaptive DFA are, first of all classified in:
  - Letters,
  - Digits
  - Spaces etc.

The process is similarly with part of the lexical analysis
Classes of characters

➲ -- Simple function to compute the class

clasa a =
    if (a >= 'a' && a <= 'z') ||
         (a >= 'A' && a <= 'Z')
    then 'l'
    else if (a >= '0' && a<= '9')
         then 'c'
    else if a == '\t' || a =='\n' || a ==' ' then '_'
    else '?'

g compelled
➲ -- 'l' = alphabetic, 'c' = digits, '_' = spaces
Preparing words for storage

- Classifying the characters from a new word means adding spaces and classify the result character after character. What we get will be called “scheme”. Ex: “_ccc_”

clasifica = (map clasa). addspace

- Where,..., you know:
- map – the usual map of functional languages
- Ex: map f [x,y,z] = [ f x, f y, f z]
- 'dot' is the product of functions
Simulating the storage in the matrix

{--

Pentru fiecare tripleta (x,y,z) de clase ale unor simboluri succeseive vom pastra schemele cuvintelor care contin acea tripleta intr-o lista asociata tripletei. Aceasta lista devine un al patrulea element.

Lista se poate afla usor filtrand dictionarul:

filter (substr (x,y,z))  dict

unde functia filtru este data de formula:
--}
Simulating the storage in the matrix

{--
For every triple \((x,y,z)\) (\(x,y,z\) being classes of successive symbols of the word) we will preserve the schemes of those words in a list which is associated with the triple, becoming the 4\(^{th}\) element.

The list can be easily found by filtering the dictionary itself:

\[
\text{filter (substr (x,y,z)) dict}
\]

where the filter is defined as:

--}
The filter

strar (x,y,z) (c1:c2:c3:t) =
if c1==x && c2==y && c3==z
then True
else strar (x,y,z) (c2:c3:t)
strar (x,y,z) (c1:c2:[]) = False

-- if the sequence of classes “xyz” is found somewhere in the scheme of the word, this fact triggers the placement in that list.
The trained Adaptive DFA

automat dict =
[ (x,y,z, filter (substr (x,y,z))  dict )
 | x <- n, y <- n , z <- n ]
where
n = "lc_"    -- n = map clasificare "D2 "

-- Note: The dict which is used here is in fact a list of schemes of the words serving as training examples.
Rebuilding examples from previous papers

Now, the adaptive DFA from [Popa05] which was trained to accept numbers can be simply defined as:

\[ a = \text{automat} ["\_c\_", "\_cc\_", "\_ccc\_"] \]

Or using examples and the classification fct.

\[ a = \text{automat} [\text{clasifica } "0", \text{clasifica } "21", \text{clasifica } "196"] \]
Rebuilding examples from previous papers

Now, the adaptive DFA from [Popa05] can be established by simply asking Hugs or GHCi to produce an explicit value:
Using a trained Adaptive DFA

 análiza cuvant automat=

 [ m | (x,y,z,m) <- automat
 , (x,y,z) `elem` triplete cuvant]

 -- and if you want to trace:

 trace cuvant automat=

 [ (x,y,z,m)| (x,y,z,m) <- automat
 , (x,y,z) `elem` triplete cuvant]
...where the proposed text is broken in “triples”...

- Auxiliary

triplete :: [Char] -> [(Char, Char, Char)]
- triplete [a, b, c] = [(a, b, c)]
- triplete (a : b : c : d) = (a, b, c) : (triplete (b : c : d))
- triplete (b : c : _) = []

Note: in the previous slide `x `elem` m` is the test “if the element x belongs to the list m“
--- Analyzing the Word using an ADFA ---

**analiza cuvant automat** = 

\[ m \mid (x,y,z,m) \leftarrow \text{automat} \]

\[ , (x,y,z) \ `\text{elem}` `\text{triplete cuvant} \]

**Remark:** The list may contains more sets of “schemes”. If one “scheme” appears in all this sets -> the word is accepted. See next slide:
Accepting a word

- Acceptance by intersection.
- When the ADFA is processing a token, it can identify more than one set of schemes partially matching that token.

acceptare cuvant automat = foldl intersect (head a) a
where
a = analiza cuvant automat
Acceptance criteria

- The intersection contains ONE or more SCHEMEs => Accepted.

- The intersection did not contain a common scheme, so it is [ ] => Not accepted.
The trained ADFA, is working now
Conclusions

➲ The adaptive automata can be build using various languages. We have tried: Oberon-2, C++, Haskell.
➲ The theory and technology may have multiple appliances: video alarm systems, automatic weapons, anti-virus products, automatic observers, music synthesis and recognition, voice identification systems...and maybe more.
Present and the next step

Testing the limits of adaptive automata:

 ➤ Compilers, interpreters, DSL's: ok, done

 ➤ Allarms triggered by image: ok, done

 ➤ Other appliances: working....
References


References


[Pop05] Popa Dan; Adaptive DFA based on array of sets, Studii si Cercetari Ştiinţifice, Seria Matematica, Nr 15 (2005) p 113-121, ISSN 1224 - 2519
References

Smeu Florin: Sistem de supraveghere video bazat pe automat adaptiv. (The student got the first prize :) ! )

http://stiinte.ub.ro/cercetare/c-conferinte/106/327