Automatic Generation of a Suffix Trie from a Naive Pattern Matcher and a Text by Partial Evaluation

Yoshihiko FUTAMURA[†] Zenjiro KONISHI[††] Kazuaki MAEHO[†††] Masahiko KAWABE[†††]

[†] School of Science and Engineering, Waseda University
[††] Institute for Software Production Technology, Waseda University
[†††] Graduate School of Science and Engineering, Waseda University

Abstract

This paper is to describe our on-going project to implement a partial evaluator to automatically generate a pattern matcher with a suffix trie from a naive pattern matcher and a given text. Partial Evaluation has been used to generate efficient string matchers from naive string matching programs. However, conventional researches have been conducted to partially evaluate a given naive matcher with respect to a given pattern. The generated programs run in O(m+n) time where *m* and *n* stand for the lengths of a pattern and a text, respectively. However, for a large text, O(n) is still expensive. Our research aims at specializing a naive matcher with respect to a text and generating an O(m)-time and O(n)-space matcher.

1. INTRODUCTION

Automatic generation of an efficient pattern matcher from a naive one, introduced in [6], is a typical problem for partial evaluation [7,8,11]. Let *m* be the length of a given pattern and *n* be the length of a given text. Then our problem can be classified as two problems:

Type 1: Can we generate an O(m) pattern matcher of size O(n) from a naive non-linear matcher and a given text?
Type 2: Can we generate an O(n) algorithm from a given matcher that generates an O(m) pattern matcher of size O(n) from a given text?

The problems can be rephrased in partial evaluation terms. Let α, *pm*, *t* and *p* be a partial evaluator, pattern matcher, text and pattern, respectively. Let the residual program of x with respect to y be x_y i.e. $x_y=\alpha(x,y)$. Then we can redefine the above problems as follows:

Type 1: Does $pm_t(p)$ run in O(m) time for any *p* and is pm_t of size O(n)?

Type 2: Does $\alpha_{pm}(t)$ run in O(n) time for any *t* ? And is $\alpha_{pm}(t)$ of size O(n)? Furthermore, does $\alpha_{pm}(t)(p)$ run in O(m) time for any *p*?

Those problems have never been solved by partial evaluation to the best of authors' knowledge. Conventional researches have been conducted to partially evaluate a given naive matcher with respect to a given pattern [1,2,3,5,9,10,15]. The generated programs run in O(m+n) time like the Knuth-Morris-Pratt [12] or the Boyer-Moore [4] pattern matcher. Apparently, Type 2 problem is more difficult than Type 1. This paper deals with Type 1 problem. When the out-put of a pattern matcher is **true** or **false**, we show that, by Generalized Partial Computation (GPC) [7,8], an O(m) pattern matcher of size O(n) can be generated from a naive non-linear matcher and a given text. This paper assumes that readers are familiar with program transformation [13] and partial evaluation [11].

2. NAIVE PATTERN MATCHER AND SUFFIX TRIE

First, we define a naive matcher nm(p,t) which searches for a pattern *p* in a text *t*. If *p* is found in *t*, *nm* returns **true**. Otherwise it returns **false** or nil, i.e. **[ ]**. The *nm* uses prefix(p,t) as an auxiliary function. The *prefix* checks if *p* is a prefix of *t*.

Definition 1: Naive Matcher
nm(p,t)≡**if** null(p) **then true**
  else if null(t) **then [ ]**
  else ∨ (nm(p,cdr(t)),prefix(p,t))
prefix(p,t)≡**if** null(p) **then true**
  else if null(t) **then [ ]**
  else if car(p)=car(t) **then** prefix(cdr(p),cdr(t))
  else [ ]

Let a given pattern be $p=a_0a_1...a_{m-1}$ and a given text be $t=t_0t_1...t_{n-1}$. We modify the above naive matcher to utilize indices of characters.

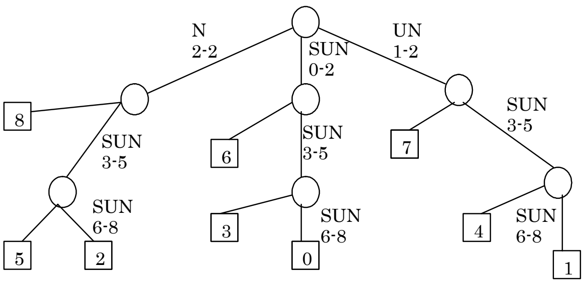
Definition 2: Modified Naive Matcher
nm(p,t)≡nm1(p,t,0)
nm1(p,t,u)≡**if** null(p) **then true**
  else if null(cd^ur(t)) **then [ ]**
  else ∨ (nm1(p,t,u+1),prefix(p,t,u))
prefix(p,t,u)≡**if** null(p) **then true**
  else if null(cd^ur(t)) **then [ ]**
  else if car(p)=car(cd^ur(t)) **then** prefix(cdr(p),t,u+1)

else [ ]

Based upon the modified naive matcher above and a given text, we generate an O(m) matcher of size O(n). The generated matcher is almost the same as the string matcher with a suffix trie described in page 399 of [14]: In an application where the text string is fixed (as for a dictionary), and many pattern lookups are to be handled, the search time can be dramatically reduced by preprocessing the text string, as follows: Consider the text string to be a set of N strings, one starting at each position of the text string and running to the end of the string (stopping *k* characters from the end, where *k* is the length of the shortest pattern to be sought). To find out whether a pattern occurs in the text, proceed down the trie from the root, going left on 0 and right on 1 as usual, according to the pattern bits. If a void external node is hit, the pattern is not in the text; if the patter exhausts on an internal node, it is in the text; and if external node is hit, compare the remainder of the pattern to the text bits represented in the external node as necessary to determine whether or not there is a match.

In this paper, we index each branch of a trie with a character in a given text.

Example 1: Trie for text SUNSUNSUN



3. PARTIAL EVALUATION

First, we define some notations for partial evaluation.

Definition 3: Notation for Partial Evaluation

Let e(u) be any expression and *i* be any information concerning *u* and auxiliary functions in *e*. Then $\vdash e(u) \dashv_i$ stands for the residual program of the partial evaluation of e(u) with respect to *i*.

Definition 4: Abbreviation for Partial Information

$f_i(x) \equiv \vdash f(x) \dashv_i$ for any function *f*. *i* can be omitted when *i* is **true** or trivial.

The problem of partial evaluation of a naive matcher with respect to a given text is that the generated matcher becomes $O(n^2)$ space because of unfolding. Therefore, we need to invent a mechanism to compress unfolded recursive calls. We just replace a sequence of tail recursive calls with a loop (Proof is omitted).

Theorem 1: Loop Introduction

Let f(x) be a tail recursive program as follows:

f(x)≡**if** c(x) **then** b(x) **else** f(d(x))

(1) b(x) does not include a recursive call to *f* .

(2) for some k>j≥0, if $c(d^j(x))$ is neither provable nor refutable but $c(d^k(x))$ is provable or refutable. Then

$\vdash f(x) \dashv_i \equiv F(f,x,0,k)$ where

F(f,x,j,k)≡**if** j<k **then** {**if** $c(d^j(x))$ **then** $\vdash b(d^j(x)) \dashv_i$ **else** F(f,x,j+1,k)}

else $\vdash f(d^k(x)) \dashv_{i \wedge \neg c(x) \wedge \ldots \wedge \neg c(d^{k-1}(x))}$

Example 2: Let *f* be a tail recursive program as follows: f(x,y,n)≡**if** x=n **then** y **else** {**if** y=0 **then** x **else** f(x+1,y-1,n)} and i=(0<n≤y), and *n* and *y* are known.

c(x,y)≡(x=n)∨ (y=0), b(x,y)≡**if** x=n **then** y **else** x. Then

$f_{y,n}(x) \equiv F(f,x,n,0,n+1)$ where

F(f,x,y,n,j,k)≡

if j<k **then**

{**if** (x+j=n)∨ (y-j=0) **then** (**if** x+j=n **then** y-j **else** x+j)

else F(f,x,y,n,j+1,k)}

else x+y.

While, $\vdash f(x) \dashv_i$ for usual partial evaluation (GPC) is:

if x=n **then** y

else if x=n-1 **then** y-1

...

else if x=0 **then** y-n

else x+y

Corollary 1: Let f(x) be a tail recursive program as follows: f(x)≡**if** c(x) **then** b(x) **else** f(d(x))

(1) b(x) does not include a recursive call to *f* .

(2) for some k>j≥0, if $c(d^j(x))$ is neither provable nor refutable but $c(d^k(x))$ is provable or refutable. Then

$\vdash f(x) \dashv_i \equiv F(f,x,0,k)$ where

F(f,x,j,k)≡**if** j<k **then** {**if** $\neg c(d^j(x))$ **then** F(f,x,j+1,k) **else** $\vdash f(d^j(x)) \dashv_{i \wedge c(x) \wedge \ldots \wedge c(d^{j-1}(x)) \wedge \neg c(d^j(x))}$}

else $\vdash f(d^k(x)) \dashv_{i \wedge \neg c(x) \wedge \ldots \wedge \neg c(d^{k-1}(x))}$

By the generalization, we transform the modified naive matcher as follows:

Definition 5: Generalization of Modified Naive Matcher

(1) h(0,p,t)=[ ]

(2) h(1,p,t,u)=prefix(p,t,u).

(3) For 1<r , 0≤ u1<u2<...<ur

h(r,p,t,u1,...,ur)≡∨ (h(r-1,p,t,u1,...,u(r-1)), h(1,p,t,ur)).

Note that nm(p,t)=h(n,p,t,0,...,n-1). By the GPC of *h*, we obtain the following theorem. The proof is given in APPENDIX 1.

Theorem 2: For 0<r , 0≤u1<u2<...<ur,

h(r,p,t,u1,...,ur)≡**if** null(p) **then true**

else if null($cd^{ur}r(t)$) **then** h(r-1,p,t, u1,...,u(r-1))

else if car(p)=car($cd^{u1}r(t)$)=...=car($cd^{ur}r(t)$) **then** h(r,cdr(p),t,u1+1,...,ur+1)

else if car(p)=car($cd^{u2}r(t)$)=...=car($cd^{ur}r(t)$) **then** h(r-1,cdr(p),t,u2+1,u3+1...,ur+1)

...

else if car(p)=car($cd^{u1}r(t)$) =...=car($cd^{u(r-2)}r(t)$)= car($cd^{ur}r(t)$) **then** h(r-1,cdr(p),t,u1+1,...,u(r-2)+1,ur)

...

else if car(p)=car($cd^{u(r-1)}r(t)$)=car($cd^{ur}r(t)$) **then** h(2,cdr(p),t,u(r-1)+1,ur+1)

...
else if car(p)=car(cd^{u1}r(t))=car(cd^{ur}r(t)) **then**
h(2,cdr(p),t,u1+1,ur+1)
else if car(p)=car(cd^{ur}r(t)) **then** h(1,cdr(p),t,ur+1)
else h(r-1,p,t,u1,...,u(r-1))

By Corollary 1, we get the Corollary 2 below.

Corollary 2:
├h(r,p,t,u1,..,ur) ┤≡F(h,r,p,t,u1,...,ur,0,k) for some k.
F(h,r,p,t,u1,...,ur,j,k)
≡**if** j<k **then** {**if** cd^{j}r(p)∧cd^{ur+j} r(t)∧car(cd^{j}r(p))=
car(cd^{u1+j}r(t))=...=car(cd^{ur+j}r(t)) **then**
F(h,r,p,t,u1,...,ur,j+1,k)
else ├h(r,cd^{j}r(p),t,u1+j,...,ur+j)┤ }
else ├h(r,cd^{k}r(p),t,u1+k,...,ur+k)┤

Example 3: Let i1≡¬null(p) and

t = (	S	U	N	S	U	N	S	U	N	)
	0	1	2	3	4	5	6	7	8	

Then, the partial evaluation (GPC) of *h* with respect to *i1* and *t* produces the following O(m) time and O(n) space matcher h(9,p,t,0,1,...,8). The derivation process is shown in APPENDIX 2. Note that h(9,p,t,0,1,...,8) has only one unknown variable *p*.

h(9,p,t,0,1,...,8)≡**if** null(p) **then true**
else if car(p)=N **then** h(3,cdr(p),t,3,6,9)
else if car(p)=U **then** h(3,cdr(p),t,2,5,8)
else if car(p)=S **then** h(3,cdr(p),t,1,4,7)
else [ ]
h(3,p,t,3,6,9)≡**if** null(p) **then true**
else h(2,p,t,3,6)
h(2,p,t,3,6)≡F(h,2,p,t,3,6,0,3)
F(h,2,p,t,3,6,j,3)
≡**if** j<3 **then** {**if** cd^{j}r(p)∧cd^{6+j} r(t)∧
car(cd^{j}r(p))=car(cd^{6+j}r(t)) **then** F(h,2,p,t,3,6,j+1,3) **else**
{**if** null(cd^{j}r(p)) **then true else** [ ]}}
else {**if** null(p) **then true else** prefix(p,t,6)}
h(3,p,t,2,5,8)≡**if** null(p) **then true**
else if car(p)=N **then** h(3,cdr(p),t,3,6,9)
else [ ]
h(3,p,t,1,4,7)≡**if** null(p) **then true**
else if car(p)=U **then** h(3,cdr(p),t,2,5,8)
else [ ]

4. CONCLUSION

We have challenged an unsolved problem of generating O(m) time and O(n) space pattern matcher by partial evaluation of a naive matcher with respect to a given text. This paper shows that we are on the midway to the goal. Our next target is to deal with a naive matcher which finds all the pattern in a text.

REFERENCES

[1] Ager, M.S., Danvy, O. and Rohde, H.K.: On Obtaining the KMP String Matcher by Partial Evaluation, Proc. of ASIA-PEPM 2002, ACM Press, September, 2002, 32-46.

[2] Ager M.S., Danvy O. and Rohde H.K.: Fast Partial Evaluation of Pattern Matching in Strings, Proc. of ACM PEPM2003, ACP Press 2003.

[3] Amtoft, T., Consel, C., Danvy, O. and Malmkjaer, K.: The abstraction and instantiation of string-matching programs, Lecture Notes in Computing Science 2566, 2002.

[4] Boyer, R.S. and Moore, J.S.: A fast string searching algorithm, Comm. ACM 20 (10) (1977) 762-772.

[5] Consel, C. and Danvy, O.: Partial evaluation of pattern matching in strings, *Information Processing Letters*, 30 (2), January 1989, 79-86.

[6] Futamura, Y. and Nogi, K.Generalized partial computation. in Bjørner, D. and Ershov, A. P. and Jones, N. D. (eds), *Partial Evaluation and Mixed Computation*, 133-151, North-Holland, 1988.

[7] Futamura, Y., Nogi, K. and Takano, A.: Essence of generalized partial computation, *Theoretical Computer Science* 90 (1991), 61-79.

[8] Futamura, Y., Konishi, Z. and Glück , R.: Program Transformation system based on Generalized Partial Computation, *New Generation Computing*, Vol.20 No.1, Nov 2001, 75-99.

[9] Futamura Y., Konishi Z., Glueck R.: Automatic Generation of Efficient String Matching Algorithms by Generalized Partial Computation, Proc.of ACM SIGPLAN ASIA-PEPM 2002, September 12-14, 2002, pp1-8.

[10] Glück R. and Klimov A.V. Occam's razor in metacomputation: the notion of a perfect process tree. In: CousotP., et al. (eds.), Static Analysis. *Lecture Notes in Comp. Science*, Vol. 724, Springer-Verlag 1993, 112-123.

[11] Jones, N. D.: An Introduction to Partial Evaluation, *ACM Computing Surveys*, Vol.28, No.3, September 1996, 480-503.

[12] Knuth, D.E., Morris, J. and Pratt, V.: Fast pattern matching in strings. *SIAM Journal on Computing*, 6(1973), 325-350.

[13] Pettorossi, A. and Proietti, M. Rules and Strategies for Transforming Functional and Logic Programs, *ACM Computing Surveys*, Vol.28, No.2, June 1996, 360-414.

[14] Sedgewick R. and Flajolet P.: *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.

[15] Sørensen M. H., Glück R., Jones N. D., A positive supercompiler. In: Journal of Functional Programming, 6(6), 1996. 811-838.

APPENDIX 1: Proof of Theorem 2

We can prove this theorem by the mathematical induction on 0<r.

(1) If r=1, then h(1,p,t,u1)≡prefix(p,t,u1)

≡**if** null(p) **then true**
else if null(cd^{u1}r(t)) **then [ ]**
else if car(p)=car(t) **then** prefix(cdr(p),t,u1+1)
else [ ]
≡**if** null(p) **then** u1
else if null(cd^{u1}r(t)) **then [ ]**
else if car(p)=car(t) **then** h(1,cdr(p),u1+1)
else h(0,p,t)
(2) If r>1, then, by the definition of *h* and the hypothesis of the induction, h(r,p,t,u1,...,ur)
≡∨ (h(r-1,p,t, u1,...,u(r-1)),prefix(p,t,ur))
≡**if** null(p) **then** ∨ (h(r-1,p,t, u1,...,u(r-1)),ur)
else if null(cd^{ur}r(t)) **then** h(r-1,p,t, u1,...,u(r-1))
else if car(p)=car(cd^{ur}r(t)) **then**
∨ (h(r-1,p,t, u1,...,u(r-1)),prefix(cdr(p),t,ur+1))
else h(r-1,p,t, u1,...,u(r-1))
≡**if** null(p) **then** ∨ (u1,...,ur)
else if null(cd^{ur}r(t)) **then** h(r-1,p,t, u1,...,u(r-1))
else if car(p)=car(cd^{u1}r(t))=...=car($cd^{u(r-1)}$r(t))
=car(cd^{ur}r(t)) **then**
∨ (h(r-1,cdr(p),t,u1+1,...,u(r-1)+1),
prefix(cdr(p),t,ur+1))
else if car(p)=car(cd^{u2}r(t))=...=car($cd^{u(r-1)}$r(t))=
car(cd^{ur}r(t)) **then**
∨ (h(r-2,cdr(p),t,u2+1,...,u(r-1)+1),
prefix(cdr(p),t,ur+1))
...
else if car(p)=car(cd^{u1}r(t))=...=car($cd^{u(r-3)}$r(t))=
car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t)) **then**
∨ (h(r-2,cdr(p),t,u1+1,...,u(r-2)+1),
prefix(cdr(p),t,ur+1))
...
else if car(p)=car($cd^{u(r-2)}$r(t))=car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t))
then∨ (h(2,cdr(p),t,u1+1,u(r-2)+1,u(r-1)+1),
prefix1(cdr(p),t,ur+1))
...
else if car(p)=car(cd^{u1}r(t))=car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t))
then ∨ (h(2,cdr(p),t,u1+1,u(r-1)+1),
prefix1(cdr(p),t,ur+1))
else if car(p)=car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t)) **then**
∨ (h(1,cdr(p),t, u(r-1)+1),prefix1(cdr(p),t,ur+1))
else if car(p)=car(cd^{ur}r(t)) **then**
∨ (h(r-2,cdr(p),t,u1,...,u(r-2)+1),prefix1(cdr(p),t,ur+1))
else h(r-1,p,t, u1,...,u(r-1))
≡**if** null(p) **then true**
else if null(cd^{ur}r(t)) **then** h(r-1,p,t, u1,...,u(r-1))
else if car(p)=car(cd^{u1}r(t))=...=car($cd^{u(r-1)}$r(t))
=car(cd^{ur}r(t)) **then** h(r,cdr(p),t,u1+1,...,u(r-1)+1,ur+1)
else if car(p)=car(cd^{u2}r(t))=...=car($cd^{u(r-1)}$r(t))=
car(cd^{ur}r(t)) **then**
h(r-1,cdr(p),t,u2+1,...,u(r-1)+1,ur+1)
...
else if car(p)=car(cd^{u1}r(t))=...=car($cd^{u(r-3)}$r(t))=
car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t)) **then**
h(r-1,cdr(p),t,u1+1,...,u(r-2)+1,ur+1)
...
else if car(p)=car($cd^{u(r-2)}$r(t))=car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t))
thenh(3,cdr(p),t,u1+1,u(r-2)+1,u(r-1)+1,ur+1)
...
else if car(p)=car(cd^{u1}r(t))=car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t))
then h(3,cdr(p),t,u1+1,u(r-1)+1,ur+1)
else if car(p)=car($cd^{u(r-1)}$r(t))=car(cd^{ur}r(t)) **then**
h(2,cdr(p),t, u(r-1)+1,ur+1)
else if car(p)=car($cd^{u(r-2)}$r(t))=car(cd^{ur}r(t)) **then**
h(2,cdr(p),t, u(r-2)+1,ur+1)
...
else if car(p)=car(cd^{u1}r(t))=car(cd^{ur}r(t)) **then**
h(2,cdr(p),t, u1+1,ur+1)
else if car(p)=car(cd^{ur}r(t)) **then** h(1,cdr(p),t,ur+1))
else h(r-1,p,t,u1,...,u(r-1))

APPENDIX 2: Derivation of h(9,p,t,0,1,...,8)

h(9,p,t,0,1,...,8)≡**if** null(p) **then true**
else if car(p)=N **then** h(3,cdr(p),t,3,6,9)
else h(8,p,t,0,...,7)
h(3,p,t,3,6,9)≡**if** null(p) **then true**
else h(2,p,t,3,6)
h(2,p,t,3,6)≡**if** null(p) **then true**
else if car(p)=S **then** h(2,cdr(p),t,4,7)
else [ ]
h(2,p,t,4,7)≡**if** null(p) **then true**
else if car(p)=U **then** h(2,cdr(p),t,5,8)
else [ ]
h(2,p,t,5,8)≡**if** null(p) **then true**
else if car(p)=N **then** h(2,cdr(p),t,6,9)
else [ ]
h(2,p,t,6,9)≡**if** null(p) **then true else** prefix(p,t,6)
├h(2,p,t,3,6) ┤≡F(h,2,p,t,3,6,0,3)
F(h,2,p,t,3,6,j,3)
≡**if** j<3 **then** {**if** cd^{j}r(p)∧cd^{6+j} r(t)∧car(cd^{j}r(p))=car(cd^{3+j}r(t))=car(cd^{6+j}r(t)) **then** F(h,2,p,t,3,6,j+1,3) **else**
├h(2,cd^{j}r(p),t,3+j,6+j)┤ }
else ├h(2,cd^{3}r(p),t,6,9)┤
≡**if** j<3 **then** {**if** cd^{j}r(p)∧cd^{6+j} r(t)∧car(cd^{j}r(p))=
car(cd^{6+j}r(t)) **then** F(h,2,p,t,3,6,j+1,3) **else**
{**if** null(cd^{j}r(p)) **then true else** [ ]}}
else {**if** null(p) **then true else** prefix(p, cd^{6}r(t))}
h(8,p,t,0,...,7)≡**if** null(p) **then true**
else if car(p)=U **then** h(3,cdr(p),t,2,5,8)
else h(7,p,t, 0,...,6)
h(3,p,t,2,5,8)≡**if** null(p) **then true**
else if car(p)=N **then** h(3,cdr(p),t,3,6,9)
else h(2,p,t,2,5)
h(2,p,t,2,5)≡**if** null(p) **then true**
else if car(p)=N **then** h(2,cdr(p),t,3,6)
else [ ]≡[ ]
h(7,p,t, 0,...,6)≡**if** null(p) **then true**
else if car(p)=S **then** h(3,cdr(p),t,1,4,7)
else h(6,p,t,0,5)
h(3,p,t,1,4,7)≡**if** null(p) **then true**
else if car(p)=U **then** h(3,cdr(p),t,2,5,8)
else h(2,p,t,1,4)
h(2,p,t,1,4)≡**if** null(p) **then true**
else if car(p)=U **then** h(2,cdr(p),t,2,5)
else [ ]≡[ ]