BRICS RS-05-14 Danvy & Rohde: On Obtaining the Boyer-Moore String-Matching Algorithm by Partial Evaluation

BRICS

Basic Research in Computer Science

On Obtaining the Boyer-Moore String-Matching Algorithm by Partial Evaluation

Olivier Danvy
Henning Korsholm Rohde

BRICS Report Series **RS-05-14**

ISSN 0909-0878 **April 2005**

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/05/14/`

On Obtaining the Boyer-Moore String-Matching Algorithm by Partial Evaluation

Olivier Danvy and Henning Korsholm Rohde
BRICS*
Department of Computer Science
University of Aarhus†

April 28, 2005

Abstract

We present the first derivation of the search phase of the Boyer-Moore string-matching algorithm by partial evaluation of an inefficient string matcher. The derivation hinges on identifying the *bad-character-shift* heuristic as a binding-time improvement, bounded static variation. An inefficient string matcher incorporating this binding-time improvement specializes into the search phase of the Horspool algorithm, which is a simplified variant of the Boyer-Moore algorithm. Combining the *bad-character-shift* binding-time improvement with our previous results yields a new binding-time-improved string matcher that specializes into the search phase of the Boyer-Moore algorithm.

*Basic Research in Computer Science (`www.brics.dk`),
funded by the Danish National Research Foundation.
†IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark.
Email: {`danvy,hense`}`@brics.dk`

Contents

1 Introduction

String matching is a traditional application of partial evaluation, and obtaining the search phases of linear-time algorithms out of inefficient string matchers has become a standard benchmark [13, 16]. The obtained algorithms include several non-trivial ones, notably the Knuth-Morris-Pratt *left-to-right* string-matching algorithm [14] and simplified variants of the Boyer-Moore *right-to-left* string-matching algorithm [5].

The Boyer-Moore algorithm uses two heuristics: *good-suffix* and *bad-character-shift*. We observe that on one hand, the simplified variants of the Boyer-Moore search phase obtained by partial evaluation use only the *good-suffix* heuristic [2, 4, 10, 11, 15], and that on the other hand, Horspool uses only the *bad-character-shift* heuristic for his own string matcher [12]. In the present work, we use both heuristics.

We follow the partial-evaluation tradition of improving the binding times of an inefficient string matcher to make it specialize to a known string matcher [8]:

1. Our first step is to express the *bad-character-shift* heuristic as a binding-time improvement in a naive, inefficient string matcher. Specializing the binding-time improved string matcher yields the search phase of the Horspool string matcher, which is a new result.

2. We then combine the *bad-character-shift* binding-time improvement with our previous results [2] and present a new binding-time-improved string matcher. Specializing this string matcher yields the search phase of the Boyer-Moore string matcher, which is our main result.

Overview: Section 2 presents the technical background: string matching, the starting inefficient string matcher, partial evaluation, and binding-time improvements. Section 3 presents the *bad-character-shift* heuristic and shows how to obtain the Horspool algorithm. Section 4 shows how to obtain the Boyer-Moore algorithm. We then address correctness issues in Section 5.

2 Preliminaries

String matching: A string-matching algorithm finds the first occurrence of a pattern string, $p = p_0 p_1 \cdots p_{m-1}$, in a text string, $t = t_0 t_1 \cdots t_{n-1}$, where strings are sequences of atomic characters of some finite alphabet, Σ. A trademark of the Boyer-Moore algorithm is to compare the characters of the pattern against the text *from right to left*, as in the following naive string matcher (adapted from our earlier work [2]):

$$
\begin{aligned}
main(p,t) &= match(p,t,|p|-1,|p|-1)\\
match(p,t,j,k) &= \text{if } \quad j=-1\\
&\quad\ \text{then } \underline{\text{match at } k+1}\\
&\quad\ \text{else } \text{if } \quad k \geq |t|\\
&\qquad\qquad \text{then } \underline{\text{no match}}\\
&\qquad\qquad \text{else } \ compare(p,t,j,k)\\
compare(p,t,j,k) &= \text{if } \quad p_j = t_k\\
&\quad\ \text{then } match(p,t,j-1,k-1)\\
&\quad\ \text{else } \text{let } offset = compute\_offset(p,t,j,k)\\
&\qquad\qquad \text{in } \ match(p,t,|p|-1,k+offset)\\
compute\_offset(p,t,j,k) &= |p|-j
\end{aligned}
$$

This program returns match at k (i.e., a result of type *int*) if the left-most occurrence of p in t begins at index k, and no match (i.e., a result of type *unit*) if p does not occur in t. We will use this program as a template for our binding-time-improved programs, modifying only the definition of *compute_offset*.

Partial evaluation: Partial evaluation is a program transformation that propagates constants, unfolds calls, and computes constant expressions [9, 13]. Its goal is to specialize programs. Given a string matcher of type $pattern \times text \rightarrow int + unit$ and a pattern string p, a partial evaluator generates a program of type $text \rightarrow int + unit$ such that for any text string t, running the source string matcher on p and t yields the same result as running the generated program on t alone.

Binding-time improvements: A binding-time improvement is a source-program transformation that makes a program specialize better [13, Chapter 12]. For example, if we assume x to be of boolean type and unknown at partial-evaluation time, we can transform the function call "$foo(x)$" into "$case\ x\ of\ true \rightarrow foo(true) \mid false \rightarrow foo(false)$," by enumerating the possible values of x. The transformation is a binding-time improvement because the argument of *foo* changes from being known only at run time (dynamic) to being known already at partial-evaluation time (static). This particular binding-time improvement—colloquially known as "The Trick"—is more descriptively referred to as "bounded static variation" nowadays [13].

Partial evaluation applied to string matching: Efficient string matchers usually consist of a pre-calculation phase (on the pattern) and a search phase (on the pattern, the result of the pre-calculation, and the text). Ideally, by specializing a string matcher with respect to a pattern, a partial evaluator computes what amounts to a pre-calculation phase and yields a specialized program that computes the search phase (on the text). A naive string matcher such as the one above, however, does not readily allow significant optimization through specialization. Successful partial evaluation of string matchers is based on the observation that after every character comparison, static information about the dynamic text must be maintained, expressed as equalities (‘$t_i = p_j$’) or inequalities (‘$t_i \neq p_j$’) with characters from the pattern. Keeping and using this information at partial-evaluation time, either by a clever partial evaluator or by a clever rewriting of the naive string matcher (i.e., a binding-time improvement), is the key to obtaining specialized programs that compute the search phase efficiently.

Challenge: Although generally successful [2, 3, 4, 8, 10, 11, 13, 15, 16], so far the program-specialization approach to string matching has failed to obtain the Boyer-Moore string matcher. This algorithm is regarded as too unsystematic to be obtainable by partial evaluation [3, 4].

3 Obtaining the *bad-character-shift* heuristic

The *bad-character-shift* heuristic improves the special case where the *first* comparison fails (which gives us only the single inequality, ‘$t_i \neq p_{|p|-1}$’). The heuristic works by taking the “bad character”, t_i, and exploiting knowledge of its last position (if any) in the pattern to safely skip a number of non-occurrence positions [5]. It works *in constant time* using a Σ-sized pre-calculated table. For example, after a mismatch, `T`$\neq$`N`, at

```
text:        "-A-TEXT-IN-WHICH-PATTERN-OCCURS-"
pattern:     "PATTERN"
                   ↑
```

the heuristic allows matching to be resumed at

```
text:        "-A-TEXT-IN-WHICH-PATTERN-OCCURS-"
pattern:       "PATTERN"
                   =  ↑
```

where the faulting `T` is safely matched.

From a partial-evaluation perspective, the key observation is that we know not only that ‘$t_i \neq p_{|p|-1}$’, but also that ‘$t_i = c_j$’ for some $c_j \in \Sigma$. Since Σ is finite, we can use bounded static variation over Σ to obtain exact static information about t_i. Continuing matching using just this piece of information turns out to precisely mimic the *bad-character-shift* heuristic, and integrating it into the inefficient string matcher of Section 2 gives a (non-trivial) binding-time-improved program. As announced in Section 2, we only modify the definition of *compute_offset*:

$$
\begin{aligned}
\mathit{compute\_offset}(p, t, j, k) &= |p| - j - 1 + \mathit{shift}(p, t_{k+|p|-j-1}) \\
\mathit{shift}(p, c) &= \text{case } c \text{ of } \begin{cases} c_1 & \rightarrow \mathit{rematch}(p, 1, c_1) \\ & \vdots \\ c_{|\Sigma|} & \rightarrow \mathit{rematch}(p, 1, c_{|\Sigma|}) \end{cases} \\
\mathit{rematch}(p, i, c) &= \text{if } \quad i = |p| \\
&\quad\ \text{then } i \\
&\quad\ \text{else if } \quad c = p_{|p|-1-i} \\
&\qquad\qquad \text{then } i \\
&\qquad\qquad \text{else } \mathit{rematch}(p, i + \mathit{1}, c)
\end{aligned}
$$

This binding-time improved, but inefficient string matcher performs the same sequence of character comparisons between the pattern and the text as the *right-to-left* variant of the Horspool variant of the Boyer-Moore algorithm [12]. Consequently, since *rematch* is static, specializing this program with respect to a pattern p (and an alphabet Σ) yields a $(p + \Sigma)$-sized program that performs identically to the search phase of the Horspool algorithm in terms of character comparisons. We assume that *case* is a constant-time primitive operation, possibly achieved separately by tabulation. The specialized program then also performs identically to the search phase of the Horspool algorithm in terms of primitive operations (modulo some arithmetic operations due to our non-optimized formulation).

For example, specializing the binding-time-improved string matcher with respect to $p =$ '*aba*' and $\Sigma = \{a, b, c\}$ yields the following program:

$$
\begin{aligned}
\mathit{main}_{aba}(t) &= \mathit{match}_{(aba,\mathit{2})}(t, 2) \\
\mathit{match}_{(aba,\mathit{2})}(t, k) &= \text{if } \quad k \geq |t| \\
&\quad\ \text{then } \underline{\text{no match}} \\
&\quad\ \text{else if } \quad 'a' = t_k \\
&\qquad\qquad \text{then } \mathit{match}_{(aba,\mathit{1})}(t, k - 1) \\
&\qquad\qquad \text{else } \mathit{match}_{(aba,\mathit{2})}(t, k + \mathit{shift}_{aba}(t_k)) \\
\mathit{match}_{(aba,\mathit{1})}(t, k) &= \text{if } \quad k \geq |t| \\
&\quad\ \text{then } \underline{\text{no match}} \\
&\quad\ \text{else if } \quad 'b' = t_k \\
&\qquad\qquad \text{then } \mathit{match}_{(aba,\mathit{0})}(t, k - 1) \\
&\qquad\qquad \text{else } \mathit{match}_{(aba,\mathit{2})}(t, k + 1 + \mathit{shift}_{aba}(t_{k+1})) \\
\mathit{match}_{(aba,\mathit{0})}(t, k) &= \text{if } \quad k \geq |t| \\
&\quad\ \text{then } \underline{\text{no match}} \\
&\quad\ \text{else if } \quad 'a' = t_k \\
&\qquad\qquad \text{then } \mathit{match}_{(aba,-\mathit{1})}(t, k - 1) \\
&\qquad\qquad \text{else } \mathit{match}_{(aba,\mathit{2})}(t, k + 2 + \mathit{shift}_{aba}(t_{k+2})) \\
\mathit{match}_{(aba,-\mathit{1})}(t, k) &= \underline{\text{match at } k + 1}
\end{aligned}
$$

$$shift_{aba}(x) = \text{case } x \text{ of } \begin{cases} a \to 2 \\ b \to 1 \\ c \to 3 \end{cases}$$

The specialized version of *shift* (i.e., $shift_{aba}$) is equivalent to the pre-calculated *bad-character-shift* lookup-table [5], in terms of both size and access time. Hence, the *bad-character-shift* heuristic, in the imperative formulation of string matching, can be viewed as an instance of bounded static variation—one that is represented efficiently.

4 From Horspool to Boyer-Moore

We can now obtain the Boyer-Moore algorithm by unifying the result from Section 3 with our earlier results on the *good-suffix* heuristic [2, Section 5]:

$$\begin{aligned}
compute_offset(p,t,j,k) &= |p| - j - 1 + \max(rematch_{gs}(p,j,|p|-1,|p|-2), \\
&\qquad\qquad\qquad\qquad shift(p,t_k) - |p| + j + 1) \\
rematch_{gs}(p,j,j',k') &= \text{if } \; k' = -1 \\
&\quad\; \text{then } j' + 1 \\
&\quad\; \text{else if } \; j = j' \\
&\qquad\quad \text{then if } \; p_{j'} \neq p_{k'} \\
&\qquad\qquad \text{then } j' - k' \\
&\qquad\qquad \text{else } rematch_{gs}(p,j,|p|-1,k'+|p|-j'-2) \\
&\qquad\quad \text{else if } \; p_{j'} = p_{k'} \\
&\qquad\qquad \text{then } rematch_{gs}(p,j,j'-1,k'-1) \\
&\qquad\qquad \text{else } rematch_{gs}(p,j,|p|-1,k'+|p|-j'-2)
\end{aligned}$$

This binding-time improved, but inefficient string matcher performs the same sequence of character comparisons between the pattern and the text as the Boyer-Moore algorithm (note that the *bad-character-shift* heuristic is used at other positions than $p_{|p|-1}$—using t_k instead of $t_{k+|p|-j-1}$—and then adjusted). Consequently, since *rematch* is static, specializing this program with respect to a pattern p (and an alphabet Σ) yields a $(2p + \Sigma)$-sized program that performs identically to the search phase of the Boyer-Moore algorithm in terms of character comparisons. Under the same assumptions as in Section 3, the specialized program also performs identically to the search phase of the Boyer-Moore algorithm in terms of primitive operations. Hence, we have obtained the Boyer-Moore string-matching algorithm by partial evaluation.

5 Correctness issues

As in our earlier work [1], we characterize a string matcher by a notion of *trace*: the sequence of character comparisons between the pattern and the text in the course of a run. Using a large test suite (several hundreds of runs), we have verified the correctness of each of our programs by automatically comparing its traces and the corresponding traces of a reference implementation [6]. A formal alternative would be to give a trace semantics to our programs and to the reference programs and to prove that they operate in lock step [1].

6 Conclusion

We have shown how to obtain the elusive search phase of the Boyer-Moore string-matching algorithm by partial evaluation of a binding-time-improved program with respect to a pattern and an alphabet. Our stepping stone has been the recognition of the *bad-character-shift* heuristic as an efficient representation of bounded static variation.

Acknowledgments: We would like to thank Mads Sig Ager for our pleasant joint initial study of partial evaluation of string matchers [1, 2]. We are also grateful to him and to Julia Lawall for comments.

This work is partially supported by the ESPRIT Working Group APPSEM II (`http://www.appsem.org`) and by the Danish Natural Science Research Council, Grant no. 21-03-0545.

References

[1] Mads Sig Ager, Olivier Danvy, and Henning Korsholm Rohde. On obtaining Knuth, Morris, and Pratt's string matcher by partial evaluation. In Chin [7], pages 32–46. Extended version available as the technical report BRICS-RS-02-32.

[2] Mads Sig Ager, Olivier Danvy, and Henning Korsholm Rohde. Fast partial evaluation of pattern matching in strings. *ACM Transactions on Programming Languages and Systems*, 2005. To appear. Available as the technical report BRICS RS-04-40. A preliminary version was presented at the 2003 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM 2003).

[3] Torben Amtoft. *Sharing of Computations*. PhD thesis, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 1993. Technical report PB-453.

[4] Torben Amtoft, Charles Consel, Olivier Danvy, and Karoline Malmkjær. The abstraction and instantiation of string-matching programs. In Torben Æ. Mogensen, David A. Schmidt, and I. Hal Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, number 2566 in Lecture Notes in Computer Science, pages 332–357. Springer-Verlag, 2002.

[5] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[6] Christian Charras and Thierry Lecroq. Exact string matching algorithms. `http://www-igm.univ-mlv.fr/~lecroq/string/`, 1997.

[7] Wei-Ngan Chin, editor. *ACM SIGPLAN Asian Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, Aizu, Japan, September 2002. ACM Press.

[8] Charles Consel and Olivier Danvy. Partial evaluation of pattern matching in strings. *Information Processing Letters*, 30(2):79–86, January 1989.

[9] Charles Consel and Olivier Danvy. Tutorial notes on partial evaluation. In Susan L. Graham, editor, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, pages 493–501, Charleston, South Carolina, January 1993. ACM Press.

[10] Yoshihiko Futamura, Zenjiro Konishi, and Robert Glück. Automatic generation of efficient string matching algorithms by generalized partial computation. In Chin [7], pages 1–8.

[11] Manuel Hernández and David A. Rosenblueth. Disjunctive partial deduction of a right-to-left string-matching algorithm. *Information Processing Letters*, 87:235–241, 2003.

[12] R. Nigel Horspool. Practical fast searching in strings. *Software—Practice and Experience*, 10(6):501–506, 1980.

[13] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, London, UK, 1993. Available online at `http://www.dina.kvl.dk/~sestoft/pebook/`.

[14] Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[15] Christian Queinnec and Jean-Marie Geffroy. Partial evaluation applied to pattern matching with intelligent backtrack. In *Proceedings of the Second International Workshop on Static Analysis WSA'92*, volume 81-82 of *Bigre Journal*, pages 109–117, Bordeaux, France, September 1992. IRISA, Rennes, France.

[16] Morten Heine Sørensen, Robert Glück, and Neil D. Jones. A positive supercompiler. *Journal of Functional Programming*, 6(6):811–838, 1996.

Recent BRICS Report Series Publications

RS-05-14 **Olivier Danvy and Henning Korsholm Rohde.** ***On Obtaining the Boyer-Moore String-Matching Algorithm by Partial Evaluation*****. April 2005. ii+8 pp.**

RS-05-13 **Dariusz Biernacki, Olivier Danvy, and Chung-chieh Shan.** ***On the Dynamic Extent of Delimited Continuations*****. April 2005. ii+32 pp. Extended version of an article to appear in** ***Information Processing Letters*****. Subsumes BRICS RS-05-2.**

RS-05-12 **Małgorzata Biernacka, Olivier Danvy, and Kristian Støvring.** ***Program Extraction from Proofs of Weak Head Normalization*****. April 2005.**

RS-05-11 **Małgorzata Biernacka, Dariusz Biernacki, and Olivier Danvy.** ***An Operational Foundation for Delimited Continuations in the CPS Hierarchy*****. March 2005. iii+42 pp. A preliminary version appeared in Thielecke, editor,** ***4th ACM SIGPLAN Workshop on Continuations*****, CW '04 Proceedings, Association for Computing Machinery (ACM) SIGPLAN Technical Reports CSR-04-1, 2004, pages 25–33. This version supersedes BRICS RS-04-29.**

RS-05-10 **Dariusz Biernacki and Olivier Danvy.** ***A Simple Proof of a Folklore Theorem about Delimited Control*****. March 2005. ii+11 pp.**

RS-05-9 **Gudmund Skovbjerg Frandsen and Peter Bro Miltersen.** ***Reviewing Bounds on the Circuit Size of the Hardest Functions*****. March 2005. 6 pp. To appear in** ***Information Processing Letters*****.**

RS-05-8 **Peter D. Mosses.** ***Exploiting Labels in Structural Operational Semantics*****. February 2005. 15 pp. Appears in** ***Fundamenta Informaticae*****, 60:17–31, 2004.**

RS-05-7 **Peter D. Mosses.** ***Modular Structural Operational Semantics*****. February 2005. 46 pp. Appears in** ***Journal of Logic and Algebraic Programming*****, 60–61:195–228, 2004.**

RS-05-6 **Karl Krukow and Andrew Twigg.** ***Distributed Approximation of Fixed-Points in Trust Structures*****. February 2005. 41 pp.**

RS-05-5 **A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations.** ***Dariusz Biernacki and Olivier Danvy and Kevin Millikin*****. February 2005.**