Computing the Border Array in Isabelle/HOL

Štěpán Holub

Department of Algebra, Faculty of Mathematics and Physics, Charles University, Sokolovská 83, 186 75 Prague

Abstract
An evaluable function computing the border array of a list in Isabelle/HOL is presented. The correctness of the function is verified in a straightforward lightweight manner, and it is applied to computation of other important properties of lists.

Keywords
border array, Isabelle/HOL, Knuth-Morris-Pratt algorithm

1. Introduction

A *border* of a word w (*word* is used as a preferred synonym for 'list' or 'string' in this text) is a word u (possibly empty, that is, of length 0) such that $u \neq w$ and u is both prefix and suffix of w. The *maximal border* of w is its longest border. The concept of the border is complementary to the concept of a *periodic root*. If $w = p \cdot u$ where u is a border of w, then p is a periodic root of w, that is, w is a prefix of $p \cdot p \cdot p \cdots$ (Here $\cdot$ denotes the concatenation, which is motivated by the fact words with concatenation form a monoid). Obviously, a border of w is determined by its length. The *border array* $\mathtt{BA}_w = [b_0, \ldots, b_{|w|-1}]$ of w is the list (of the same length as w, which is denoted here by $|w|$) of natural numbers such that b_i is the length of the maximal border of the prefix of w of length $i + 1$. In particular, the last element of $\mathtt{BA}_w$ is the length of the maximal border of w.

The border array, possibly slightly modified and with different terminology, is a well known structure. In particular, the border array of the searched pattern plays a crucial role in the Knuth-Morris-Pratt text search algorithm (cf. the function f defined on p. 327 of [1]). Moreover, establishing the maximal border is itself a search task: the maximal border of w corresponds to (the beginning of) the first repeated occurrence of w in itself. It follows that using the efficient Knuth-Morris-Pratt algorithm for the computation of the maximal border of a word is equivalent to the first part of the algorithm, namely to the computation of the whole border array.

2. Motivation

My interest in formalization of different aspects of periods of words, and therefore in (maximal) borders, is motivated by the project formalizing combinatorics on words, see [2] and [3]. The maximal border can be used for characterization of several other important properties od the

FMM 2021 – Fifth Workshop on Formal Mathematics for Mathematicians at CICM 2021, 30—31 July 2021
holub@karlin.mff.cuni.cz (Š. Holub)
0000-0002-6169-5139 (Š. Holub)

word, for example for establishing its primitivity. The word is *primitive* if it is its own (trivial) power only. In other words, a word is *imprimitive* if it is a power of a shorter word. For example, $abab$ is imprimitive, while $ababa$ is primitive. Denote the shortest periodic root of w by $\pi(w)$. That is, $\pi(w)$ is the shortest word such that w is a prefix of $\pi(w) \cdot \pi(w) \cdot \pi(w) \cdots$. The following lemma holds:

Lemma 1. *The word w is imprimitive if and only if $\pi(w) \neq w$ and $\pi(w) \cdot w = w \cdot \pi(w)$.*

Proof sketch. Let $w = r^k$. Then both r and $\pi(w)$ are periods of w, which implies $|\pi(w)| \leq |r|$ by the minimality of $\pi(w)$. The proof is based on the fact (which is a weak version of the Periodicity Lemma, see [4])) that if $|r| + |\pi(w)| \geq |w|$, then r and $\pi(w)$ are powers of the same word (which is then also a periodic root of w). This implies that $\pi(w) = r$ if $k \geq 2$.

On the other hand, words commute if and only if they are powers of the same word. The rest is easy. ☐

If we denote the maximal border of w by $\beta(w)$, we have $w = \pi(w) \cdot \beta(w)$. Therefore, being able to compute the maximal border yields an effective test of the primitivity of the word.

3. The algoritm

For convenience and future reference, I briefly and informally review a concrete version of the well known algorithm computing the border array. Let w be the word whose maximal border we want to compute. Fix the elements of w as $w = [w_n, w_{n-1}, \ldots, w_1, w_0]$, where $|w| = n + 1$. Following the order of construction of lists we shall process w from right to left, and construct gradually its *suffix border array*, that is, a list $[b_{|w|}, b_{|w|-1}, \ldots, b_1]$ of natural numbers where b_i is the length of the maximal border of s_i, which is the suffix of w of length i.

The construction uses, in addition to w, three variables $\mathtt{arr}$, $\mathtt{pos}$ and $\mathtt{bord}$ with the following meaning. The integer $\mathtt{pos}$ indicates the currently processed position of w, while $\mathtt{arr}$ is the suffix border array of the already processed part. More precisely, we have $w = w_1 \cdot w_2$, where $\mathtt{pos} = |w_1|$ and $|\mathtt{arr}| = |w_2|$. The list $\mathtt{arr} = [b_{|w_2|}, \ldots, b_1]$ of natural numbers is the suffix border array of w_2 (and a suffix of the computed suffix border array of w). The initial value of $\mathtt{bord}$ (for a new $\mathtt{pos}$) is $b_{|w_2|}$.

The algorithm terminates with $\mathtt{pos} = 0$. If $\mathtt{pos} \neq 0$, let a be the last element of w_1. The algorithm is currently looking for the maximal border of $a \# w_2$ (where $\#$ denotes insertion of an element at the beginning of a list), and considers $a \# p$ as a candidate, where p is the prefix of w_2 of length $\mathtt{bord}$. Moreover p (or $\mathtt{bord}$) has two additional properties:

- p is a border of w_2;
- $\mathtt{bord} + 1$ is an upper bound on the length of the maximal border of $a \# w_2$.

These conditions reflect the basic idea of the algorithm: if $a \# u$ is the maximal border of $a \# w_2$, then u is a border of w_2, and (in view of the second condition) also a border of p, which is also a suffix of w_2. This dictates the next step: the algorithm compares a and $w_{\mathtt{bord}}$.

- If $a = w_{\mathtt{bord}}$, then $a \# p$ is the maximal border of $a \# w_2$. Then $\mathtt{arr}$ can be updated, and the algorithm proceeds to $\mathtt{pos} - 1$;

- If $a \neq w_{\texttt{bord}}$, then there are two possibilities:
  - if p is empty, then $a\#w_2$ is unbordered (its maximal border is empty), and we can again proceed to $\texttt{pos} - 1$;
  - otherwise, the next candidate is $a\#p'$ where p' is the maximal border of p. The length of p' is stored in arr, namely, it is $b_{\texttt{bord}}$.

4. Implementation and related work

My formalization of the above described algorithm in Isabelle/HOL is realized by three functions: kmp_arr, kmp_bord and kmp_pos which map the four parameters w, arr, bord and pos to updated values corresponding to the situation after one step of the algorithm. The algorithm kmp is then a straightforward recursive call which is exited when $\texttt{pos} = 0$. The termination is obtained easily by the lexicographic order on the pair $(\texttt{pos}, \texttt{bord})$.

The invariant properties of variables described above are captured by the predicate kmp_valid. The key task is then to prove lemma kmp_valid_step which shows that the properties are indeed preserved by the above mentioned functions. That is, if $(w, \texttt{arr}, \texttt{bord}, \texttt{pos})$ satisfies the predicate, then also the quadruple

$$\big(w,\ \texttt{kmp\_arr}(w\,\texttt{arr}\,\texttt{bord}\,\texttt{pos}),\ \texttt{kmp\_bord}(w\,\texttt{arr}\,\texttt{bord}\,\texttt{pos}),\ \texttt{kmp\_pos}(w\,\texttt{arr}\,\texttt{bord}\,\texttt{pos})\big)$$

does. A fully structured commented proof in Isar language is given. Together with obvious validity of the predicate for the initial quadruple $(w, [0], 0, |w| - 1)$, this establishes the correctness proof, which is explicitly reformulated for the border_array (which simply inverts the suffix border array yielded by kmp).

Summarizing, our main goal is achieved by a pair of theorems. The correctness theorem bord_array shows that the function border_array indeed computes the desired length of maximal borders:

theorem bord_array: **assumes** Suc k ≤ |w|
shows (border_array w)!k = |max_border (take (Suc k) w)|

The function can be evaluated. For example,

value border_array [5,4,5,3,5,5,5,4,5::nat]

yields

[0, 0, 1, 0, 1, 1, 1, 2, 3]

This trivially leads to the code equation generating theorem max_border_comp that computes the maximal border of w.

theorem max_border_comp [code]: max_border w = take ((border_array w)!(|w|−1)) w

As can be seen, the described formalization is a handmade version of a verification process which can be nowadays heavily automatized by the Isabelle Refinement Framework (IRF) by Peter Lammich (see [5], [6], [7]). Moreover, this very framework has been used to formalize the full Knuth-Morris-Pratt algorithm by Fabian Hellauer [8]. Specifically, our function border_array (which is essentially the above mentioned function f of [1]) is defined (again in a slightly modified form) in [8] via its specification, computed by means of the Refinement Framework, and its correctness is then proved using tools of refinement automation (for example 18 gooals out of 22 in the correctness proof are discharged by the method `vc_solve` taking several seconds). The theory [8] is well written, and it contains, unsurprisingly, many theorems about borders that closely match some of my theorems and that I could easily reuse.

Therefore, in view of the above work, the value of the formalization presented here is open to judgment. I want to make two comments. First, using IRF brings about a significant overhead (in terms of the bootstrapping time for example) which discourages me from making my theory depend on it. In particular since the theory of borders is relatively simple layer of our larger development. Second, it is not clear (to me) how easily the exported code can be used in subsequent proofs in the way we need as indicated in Section 2, and described in the next section.

5. Towards the intended application

Using appropriate code equations we are now able to evaluate several important properties of particular words, like the maximal border, the minimal periodic root and the minimal period, as well as predicates `bordered` and `primitive`. For example, primitivity can be tested using Lemma 1 and the equality $w = \pi(w) \cdot \beta(w)$ by the following four pieces of code leading to the function KMP described in the previous section:

lemma primitive_iff [code]: primitive w $\longleftrightarrow$ w $\neq \varepsilon \wedge$ (π w = w $\vee$ π w $\cdot$ w $\neq$ w $\cdot$ π w)

lemma min_per_root_take [code]: π w = take (|w| − |max_border w|) w

lemma max_border_comp [code]: max_border w = take ((border_array w)!(|w|−1)) w

fun border_array :: ′a list ⇒ nat list **where**
border_array $\varepsilon = \varepsilon$
| border_array (a#w) = rev (KMP (rev (a#w)) [0] 0 (|a#w|−1))

An obvious drawback is that the evaluation is available only for types that are of class `equal`, which does not apply to general lists of type `'a list`. In order to avoid this limitation, we encode w into a binary alphabet (which is of class `equal`) by a simple function `bin_encode`:

fun bin_encode :: ′b ⇒ ′b ⇒ ′b ⇒ Enum.finite_2
where bin_encode x y = (λ z. (if z = x then Enum.finite_2.a_1 else Enum.finite_2.a_2))

Decoding function `bin_decode` is analogous. We can now prove that the encoded word is primitive if and only if the original one is:

lemma prim_bin_enc_iff: **assumes** x $\neq$ y **and** ws $\in$ lists {x,y}
shows primitive ws $\longleftrightarrow$ primitive (map (bin_encode x y) ws)

Theories

An archive version of the formalization described in this paper is available at [9] and consists of four theories:

- **CoWBasic.thy** introduces large amount of properties of words extending further the deafult theory List.thy and its extension Sublist.thy in Isabelle's HOL-Library.

- **Reverse_Symmetry.thy** is an auxiliary theory for CoWBasic.thy automating generation of reverse-symmetrical facts.
- **Border_Array.thy** contains the main material described in this paper.
- **Spehner.thy** illustrates a particular use of the primitivity predicate and the encoding into the binary alphabet.

A current (and possibly significantly modified) version of these theories is maintained as part of our large development [3].

Note added in proof: Recently, we implemented an alternative method of proving primitivity of a word, which makes this particular application of the border array calculation slightly outdated. We have also already formalized the theorem `spehner`, which was used without proof (with `sorry`) for sake of illustration in the theory **Spehner.thy** (see [10]).

Acknowledgements

The author acknowledge support by the Czech Science Foundation grant GAČR 20-20621S.

References

[1] D. E. Knuth, J. James H. Morris, V. R. Pratt, Fast pattern matching in strings, SIAM Journal on Computing 6 (1977) 323–350. doi:10.1137/0206024.

[2] Š. Holub, M. Raška, Š. Starosta, Combinatorics on words basics, Archive of Formal Proofs (2021). https://isa-afp.org/entries/Combinatorics_Words.html, Formal proof development.

[3] Š. Holub, Š. Starosta, et al., Combinatorics on words formalized, https://gitlab.com/formalcow/combinatorics-on-words-formalized, 2021.

[4] N. J. Fine, H. S. Wilf, Uniqueness theorems for periodic functions, Proceedings of the American Mathematical Society 16 (1965) 109–109. URL: http://dx.doi.org/10.1090/S0002-9939-1965-0174934-9. doi:10.1090/S0002-9939-1965-0174934-9.

[5] P. Lammich, The imperative refinement framework, Archive of Formal Proofs (2016). https://isa-afp.org/entries/Refine_Imperative_HOL.html, Formal proof development.

[6] P. Lammich, Refinement to imperative/hol, in: C. Urban, X. Zhang (Eds.), Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings, volume 9236 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 253–269. URL: https://doi.org/10.1007/978-3-319-22102-1_17. doi:10.1007/978-3-319-22102-1_17.

[7] P. Lammich, Refinement based verification of imperative data structures, in: J. Avigad, A. Chlipala (Eds.), Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016, ACM, 2016, pp. 27–36. URL: https://doi.org/10.1145/2854065.2854067. doi:10.1145/2854065.2854067.

[8] F. Hellauer, P. Lammich, The string search algorithm by Knuth, morris and pratt, Archive of Formal Proofs (2017). https://isa-afp.org/entries/Knuth_Morris_Pratt.html, Formal proof development.

[9] Š. Holub, Š. Starosta, et al., Combinatorics on words formalized, https://gitlab.com/formalcow/combinatorics-on-words-formalized/-/tree/Archive-Border-Array, 2021.

[10] Š. Holub, M. Raška, Š. Starosta, Binary codes that do not preserve primitivity, IJCAR 2022, accepted.