2 Pre-processing signatures

2.1 Background for pattern matching algorithm

The pattern matching algorithm is based on the Boyer-Moore-Horspool (BMH) algorithm for string matching (Horspool, 1980). The BMH algorithm does not allow for wildcard characters such as defined above, and so a certain amount of pre-processing needs to be performed on the internal signatures before the algorithm can be applied. The signatures are stored in unprocessed form in the PRONOM registry, and pre-processing is automatically applied as part of the generation of a new signature file.

2.2 Pre-processing of the signature file

The following pre-processing is required on each byte sequence to be used in the pattern matching. To simplify the exposition, we will use the following sequence, defined with an offset of 10 bytes from the beginning of the file as a running example:

```
A1A2A3[A4:A5]??B1B2B3(B4|B5)*{5}01??C1C2C3{4-7}D1????F1(F2|F3)F4F5
```

2.2.1 Step 1. Split the byte sequence pattern into fragments to remove all '*'

- Split up each byte sequence pattern P into smaller fragments (P_1 - P_n) wherever a "*" is found (assume P_1 – P_n are in order left to right). Drop any wildcards of the form {n} or {n-m} which appear at the ends of the P_i.

In the example:

P_1 = A1A2A3[A4:A5]??B1B2B3(B4|B5)

P_2 = 01??C1C2C3{4-7}D1????F1(F2|F3)F4F5 (note that we have dropped the "{5}")

2.2.2 Step 2. Find the minimum and maximum subsequence offsests

- If the pattern is **not** defined relative to EOF:
  - For each sequence P_i, work out the minimum and (for P_1) maximum distance of the start of P_i from the end of P_{i-1} (or the start of the file, for P_1).
- Alternatively, if the pattern **is** defined relative to EOF:
  - For each sequence P_i, work out the minimum and (for P_n) maximum distance of the end of P_i from the start of P_{i+1} (or the end of the file, for P_n).

In the example:

P_1 has minimum and maximum subsequence offsets of 10 (recall that the pattern was defined with an offset of 10 bytes from BOF).

P_2 has a minimum subsequence offset of 5 (from the “{5}” we dropped).

2.2.3 Step 3. Find the longest unambiguous byte sequence in every fragment

- For each pattern fragment, pull out longest unambiguous byte sequence (i.e. not containing ??, {n}, {k-m}, (a|b), [!a:b]): P_1X – P_nX (if there is more than one possible longest byte sequence for P_iX, choose one arbitrarily.)
- If the pattern is **not** defined relative to EOF:
    - Work out the minimum offset of the start of P_iX from the start of P_i. This is the “minimum fragment length”.
- Alternatively, if the pattern **is** defined relative to EOF:
    - Work out the minimum offset of the end of P_iX from the end of P_i. This is the “minimum fragment length”.

In the example:

P_1X = B1B2B3 (although it could equally well have been A1A2A3). The minimum fragment length is 5 (the length of “A1A2A3[A4:A5]??”).

P_2X = C1C2C3. The minimum fragment length is 2 (the length of “01??”).

2.2.4 Step 4. Split the fragments into remaining unambiguous byte sequences

- For each P_i, split up the remainder of the sequence (i.e. the part not in P_iX) to the left and the right of P_iX according to any occurrences of ? or {n} or {k-m}. This creates arrays of objects P_iL_j (to the left of P_iX) and P_iR_j (to the right of P_iX) where each of these objects contains one or more unambiguous sequences (i.e. if “|” occurs in sequence, then list all possibilities). Unlike in the previous step, these sequences may contain occurrences of the [:] wildcards, and are therefore not technically unambiguous.
- For each subsequence P_iL_j, calculate the minimum and maximum offsets of the end of P_iL_j from the start of P_iL_{j+1} (or the start of P_iX, for the rightmost P_iL_j).
- Similarly, for each subsequence P_iR_j, calculate the minimum and maximum offsets of the start of P_iR_j from the end of P_iR_{j-1} (or the end of P_iX, for the leftmost P_iR_j).

In the example:

P_1L_1 = A1A2A3[A4:A5], with a minimum and maximum offset of 1 (the "??")

P_1R_1 = B4 or B5, with a minimum and maximum offset of 0 (since P_1R_1 follows directly on from P_1X).

P_2L_1 = 01, with a minimum and maximum offset of 1 (the "??")

P_2R_1 = D1, with a minimum offset of 4, and a maximum offset of 7 (corresponding to "{4-7}").

P_2R_2 = F1F2F4F5 or F1F3F4F5, with a minimum and maximum offset of 2 (corresponding to "????").

2.2.5 Step 5. Calculate the 'shift distance': the minimum distance between each byte and the end (or start) of the longest unambiguous byte sequence in its fragment.

- For each distinct byte, *b*, the "shift distance" $D_i(b)$ is equal to the minimum distance from the end of pattern P_iX to the occurrence of that byte in P_iX (unless the byte sequence is defined relative to EOF, in which case it is from the start of P_iX). Any bytes which do not occur in P_iX are given a shift distance equal to the length of P_iX + 1.

In the example, the shift distances for P_1 are given by:

D_1(B3) = 1

D_1(B2) = 2

D_1(B1) = 3

D_1(*<all other bytes>*) = 4

The shift distances for P_2 are defined similarly.

2.3 Pre-processing glossary

Term	Meaning
P	A byte sequence pattern as contained in the internal signature
P_i	A byte sequence pattern fragment created by splitting the pattern so that it does not contain a '*'
P_iX	The longest P_i in P.
Minimum fragment length for P_iX	If the pattern is **not** defined relative to EOF, this is the minimum offset of the start of P_iX from the start of P_i. If the pattern **is** defined relative to EOF, this is the minimum offset of the end of P_iX from the end of P_i.
P_iL_j	A byte sequence pattern fragment to the left of P_iX
P_iR_j	A byte sequence pattern fragment to the right of P_iX
Minimum and maximum offsets	For each subsequence P_iL_j, these are the minimum and maximum number of bytes between the end of P_iL_j from the start of P_iL_{j+1} (or the start of P_iX, for the rightmost P_iL_j). Similarly, for each subsequence P_iR_j, these are the minimum and maximum number of bytes between the start of P_iR_j from the end of P_iR_{j-1} (or the end of P_iX, for the leftmost P_iR_j).
D_i(*byte*)	The 'shift distance' of that byte. This is defined as the minimum distance from the end of pattern P_iX to the occurrence of that byte in P_iX (unless the byte sequence is defined relative to EOF, in which case it is from the start of P_iX).

3 The pattern matching algorithm

The direction in which the pattern matching is carried out is determined by whether the byte sequence is relative to BOF, EOF or neither. The default in the following algorithm is that it is carried out from left to right from the beginning of the file, but if byte sequence is relative to EOF, then the pattern matching will be carried out from right to left, starting at the end of the file. The latter is described below by text in brackets: (/…).

1. We begin by trying to find P_1X (/P_nX). To this end, commence the search at the beginning (/end) of file F, at an offset of the minimum subsequence offset plus the minimum fragment length. This is the earliest (/latest) point in the file at which P_1X (/P_nX) may occur. Take a "window" on F of the same length as P_1X (/P_nX) and compare it with sequence P_1LX (/P_nX).

2. If the window and sequence don't match, then get the "shift distance" associated with the first byte to the right (/left) of the window in F. Shift the window forwards (/backwards) by that many bytes.

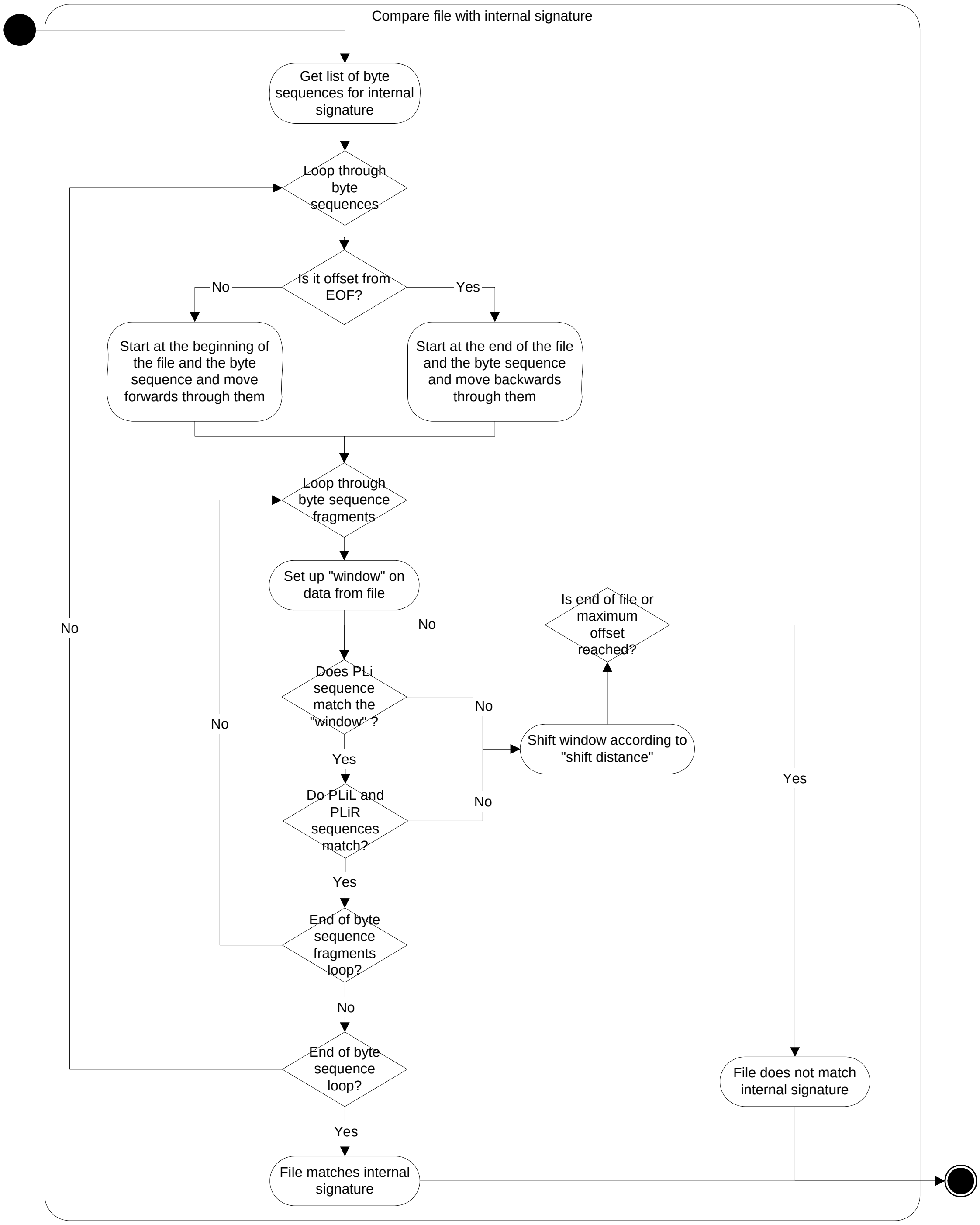
3. Now repeat step 2 until either a match is found (move on to next step) or until either the end (/beginning) of the file or the maximum offset is reached (byte sequence fails).

4. Check for matches to the P_1L_i and P_1R_i (/P_nL_i and P_nR_i) to see whether a match for the whole of P_1 (/P_n) can be found. For each sequence, start from smallest possible offset and stop as soon as a match is found so that we end up with the shortest possible sequence in F that matches the pattern. If matches are found for all these sequences, then record the location of the rightmost (/leftmost) byte for match of P_1 (/P_n) as this will determine where the search for the next pattern fragment starts. If no match is found for P_1 (/P_n), then search for next possible occurrence of P_1X (/P_nX) as in steps 2 and 3 until either the end of the file or the maximum offset is reached.

5. Now that we have found a match for the whole of P_1, repeat steps 1 to 4 for the remaining patterns P_2 to P_n. In each case however, do not begin searching at the start (/end) of the file, but at the rightmost (/leftmost) byte of the last pattern found, as recorded in Step 4 (plus (/minus) the minimum subsequence offset for the pattern). Continue until all patterns have been found (i.e. positive match) or until the end (/start) of the file is reached (i.e. no match).

Note that because we search for the patterns in order (from left to right, in the default case), and always find the earliest occurrence of each pattern, there is never any need to backtrack using this algorithm. If we can't find pattern P_3, say, we know that this has nothing to do with the placement of patterns P_1 and P_2.

An activity diagram for the pattern matching is given below (this corresponds to the 'comparison of the file with an internal signature' step in the main activity diagram in section 3.1).


Compare file with internal signature
Get list of byte sequences for internal signature
Loop through byte sequences
Is it offset from EOF?
No
Yes
Start at the beginning of the file and the byte sequence and move forwards through them
Start at the end of the file and the byte sequence and move backwards through them
Loop through byte sequence fragments
Set up "window" on data from file
No
Is end of file or maximum offset reached?
Does PLi sequence match the "window" ?
No
Shift window according to "shift distance"
Yes
Do PLiL and PLiR sequences match?
No
Yes
No
End of byte sequence fragments loop?
No
End of byte sequence loop?
No
Yes
File matches internal signature
Yes
File does not match internal signature