JFST Binary Format Description

Thora Daneyko

June 26, 2018

1 Description

First, the symbols of the alphabet are enumerated. Each character is represented by their 16-bit Java representation. Since JFST allows multichar symbols, the end of a symbol is marked by a newline character. Each symbol implicitly receives an integer id in the order in which they are enumerated, starting from 0 (the first symbol is symbol 0, the next is symbol 1, etc.). The end of the alphabet is marked by an extra newline character (i.e. two newline characters in a row $\rightarrow$ end of alphabet).

Next comes the number of states and the number of transitions, each written as a 32-bit integer.

Afterwards, the states and their transitions will be listed. Again, each state implicitly receives an id. The first state listed is the start state and receives the id 0. Literal transitions are stored in the format <to-state id + in-sym id + out-sym id>. The most significant bit of each state and symbol id is 0 to distinguish them from certain control bytes that start with 1. The end of the list of literal transitions is marked by the control byte 1000000x, where x is 1 if the state is accepting and 0 if not. Afterwards, the to-states of the identity transitions of the state are listed. The end of a state is marked by the control byte 11111111.

To save space, the length of the state and symbol ids in the transitions depends on the total number of states and symbols. Each id is encoded with the minimum number of bytes required to store the highest id (with a leading 0). I.e. a transducer with up to 128 states only needs one byte per state id (the highest state id being 127 or 01111111 which is encodable in one byte with a leading 0), but a transducer wit 129 states would require two bytes per state id (the highest state id being 128 which needs two bytes, 00000000 10000000, to have a leading 0).

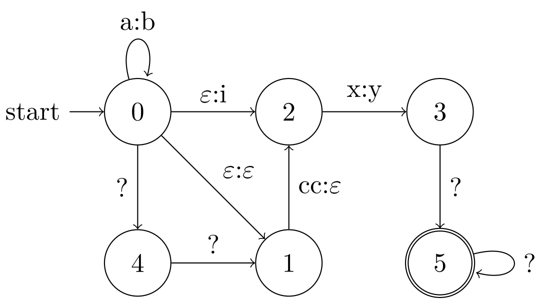

Figure 1: A simple finite state transducer.

2 Example

Figure 1 shows a very simple FST with a number of literal, epsilon and identity transitions and a multichar symbol (cc). This is what the binary file of that transducer looks like (in hexadecimal):

```
00 61 00 0A 00 62 00 0A 00 00 00 0A 00 69 00 0A 00 63 00 63 00 0A 00
78 00 0A 00 79 00 0A 00 0A 00 00 00 06 00 00 00 09 02 02 03 01 02 02 00
00 01 80 04 FF 02 04 02 80 FF 03 05 06 80 FF 80 05 FF 80 01 FF 81 05
FF
```

It can be parsed as follows:

00 61 00 0A	`a\n` / symbol 0 = a
00 62 00 0A	`b\n` / symbol 1 = b
00 00 00 0A	`<NUL>\n` / symbol 2 = ε
00 69 00 0A	`i\n` / symbol 3 = i
00 63 00 63 00 0A	`cc\n` / symbol 4 = cc
00 78 00 0A	`x\n` / symbol 5 = x
00 79 00 0A	`y\n` / symbol 6 = y
00 0A	`\n` / end of alphabet (7 symbols → 1 byte per symbol id)
00 00 00 06	6 states (→ 1 byte per state id)
00 00 00 09	9 transitions; begin state 0
02 02 03	ε:i to state 2
01 02 02	ε:ε to state 1
00 00 01	a:b to state 0
80	non-accepting, begin identity transitions
04	? to state 4
FF	end of state; begin state 1
02 04 02	c:ε to state 2
80 FF	non-accepting; (no identity transitions); end of state; begin state 2
03 05 06	x:y to state 3
80 FF	non-accepting; (no identity transitions); end of state; begin state 3
80	non-accepting, begin identity transitions
05	? to state 5
FF	end of state; begin state 4
80	non-accepting, begin identity transitions

01	? to state 1
FF	end of state; begin state 5
81	accepting, begin identity transitions
05	? to state 5
FF	end of state