FSASIM: A Simulator for Finite-State Automata

P. N. Hilfinger

1 Overview

The **fsasim** program reads in a description of a finite-state recognizer (either deterministic or non-deterministic), and a sequence of strings. It reports which strings are recognized.

For example, here is a description of a (deterministic) FSR for the language described by the regular expression '`(01)*00`':

```
alphabet [01]
start state Begin
  [0] -> Zero
state Zero
  [0] -> Done
  [1] -> Begin
final state Done
```

The `alphabet` declaration describes the alphabet used by the machine, in the format used by the **lex** program to denote a pattern that matches one of a set of single characters.

The `start state`, `state`, and `final state` declarations introduce new states. Use `start state` to introduce the initial state (by default, the first state declared). Use `final state` to introduce a final state.

The lines containing `->` denote transitions out of the last state declared. Each transition has the form of a character-set pattern as in **lex** followed by an arrow, followed by a destination state.

Here is another example, this time for Ada identifiers, which start with a letter, followed by letters, digits, and underscores, but not containing adjacent underscores and not ending with an underscore:

```
alphabet [a-zA-Z0-9_]

start state 0
   [a-zA-Z] -> 1
final state 1
   [a-zA-Z0-9] -> 1
   [_] -> 2
state 2
   [^_] -> 1
```

As in **lex**, the notation '`[^...]`' used under state 2 means "any character in the alphabet other than. . . ."

Here is an example of a non-deterministic machine's description.

```
# Recognizer for 0(01)*1|(0|1)*0
alphabet [01]
start state A
  [0] -> B
  [01] -> A
  [0] -> C
final state C
state B
```

```
  [0] -> B1
  [1] -> D
state B1
  [1] -> B
final state D
```

The following alternative machine description recognizes the same language, and uses epsilon transitions.

```
# Recognizer for 0(01)*1|(0|1)*0
alphabet [01]
start state Init
  -> A1 # Epsilon transition
  -> B1 # Epsilon transition
state A1
  [0] -> A2
  [1] -> A1
state A2
  -> F
  -> A1
state B1
  [0] -> B2
state B2
  -> B4
  [0] -> B3
state B3
  [1] -> B2
state B4
  [1] -> F
final state F
```

After reading in the machine description, **fsasim** will process any number of quoted input strings from a file or the standard input. For the machine just above, one might have the following session:

```
% fsasim desc.fsm sample.inp
"" rejected.
"001010" accepted.
"001011" accepted.
"110001" rejected.
"110000" accepted.
```

where `sample.inp` contains

```
""
"001010"
"001011"
"110001"
"110000"
```

2 FSA Descriptions

FSA descriptions are in free format; that is, you may insert additional whitespace (blanks, tabs, newlines) freely except in the middle of keywords or charsets (described below). You may also insert comments between declarations (but not within them). A comment starts with the character '`#`' and proceeds to the end of the line.

An FSA description has the following format

- An optional **alphabet** declaration of the form

  `alphabet` *charset*

  (See the description of charsets below). If this declaration is absent, the alphabet is taken to consist of all characters between blank and delete (i.e., between ASCII codes 32 and 127), inclusive.
- Any number of state declarations, consisting of
  - A header line having one of the three forms

    `state` *name*
    `start state` *name*
    `final state` *name*

    denoting a state labeled *name* that is, respectively, an ordinary state, the starting state (there should be only one), or a final state (of which there may be any number). The label *name* may be any sequence of letters, digits, periods, hyphens, underscores, and dollar signs.
  - Zero or more transition declarations having one of the two formats

    charset `->` *state*
    `->` *state*

    Denoting, respectively, a transition on any of the characters in *charset* or an epsilon transition.

Charsets

Ordinary transitions and the `alphabet` declaration use *charsets* to describe sets of characters. These consist of a sequence of characters or character ranges surrounded in square braces (`[]`). If the first character is '`^`', the charset denotes the *complement* of the character set specified by the remaining characters, relative to the declared alphabet (this form may not be used in the `alphabet` declaration itself). You may use any characters in a charset, but to include the special characters '`]`', '`\`', '`^`', or '`-`', you should precede them with a backslash ('`\`'). You can use the standard C escape characters for tabs, newlines, returns, etc.

You may denote a range of characters as '*c1*`-`*c2*', which is the same as listing all characters whose codes are greater than or equal to that of *c1* and less than or equal to that of *c2*.

3 Using fsasim

To invoke **fsasim** from a shell, type

`fsasim [-v] [--verbose] [--deterministic] [--limit=`*limit*`] [`*inputfile* `...]`

The first *inputfile* contains the machine description. In the absence of an *inputfile*, all input comes from the standard input (which is also denoted by an *inputfile* of `-`).

The input strings to be tested may either be appended to the FSA description, following the keyword `input`, placed in subsequent *inputfiles*, or (if there is at least one *inputfile*) in the standard input.

Each input string is surrounded in double quotes. Any embedded double quotes must be preceded by a backslash ('\'). You may also use the usual C/C++ conventions for special characters (e.g., '`\n`' for newline). You may optionally use whitespace to separate input strings.

The –**deterministic** option causes **fsasim** to reject any non-deterministic machine description. The –**limit** option issues a warning if the machine has more than *limit* states. The –**verbose** option causes **fsasim** to print out various explanatory messages. Finally, **-v** simply prints the **fsasim** version number and exits.