Chapter 1

FstStudio User Manual

1.1 Introduction

FstStudio (Finite State Transducer Studio) is a program for construction and running of finite state transducers. If you're not familiar with the concept of transducers and regular relations, then Xerox's fst homepage
http://www.xrce.xerox.com/research/mltt/fst/
is recommended as an introduction (primarily because the syntax of Xerox's fst program has functioned as an inspiration for the syntax of fstStudio).

1.2 Installation

If you have access to the binary file, then all you have to do is to copy the file to an appropriate library and run the program according to the instructions below. If you have access to the source code, compile the system by typing "make". This will construct a file named "fst" in the source library. To be able to compile the system, you must have ghc installed at your system. If you don't have it already, a free distribution of ghc can be downloaded at http://www.haskell.org/ghc/. It can also be run in Hugs. Load 'Main.hs' and type 'main'.

1.3 Syntax

fstStudio takes a program consisting of regular relations that denotes the relation between two regular languages and constructs a transducer. If a regular expression, not a relation, is given, then it is interpreted as the identity relation. The syntax is very similar to Xerox's finite state transducer syntax with two fundamental differences: a distinction is made between functions (definitions) and strings, and fststudio allows functional definitions.

"a"

symbol.
Example: ["b"] denotes the language {"b"}.

a

variable. A symbol without quotes is a variable.

"a":"b"

Describes a relation between the symbol a and b. This relation is ordered and a is said to be a part of the upper language and b is said to be part of the lower language.
Example: ["a":"b"] denotes the relation {("a","b")}.

0

epsilon symbol. The epsilon symbol denotes the string with no symbols.
Example: [0] denotes the language {""}.

?

all symbol. The all symbol denotes the union of all symbols in the alphabet.
Example: [?] and an alphabet {a, b, c} denotes the language {"a","b","c"}.

""

quotes cancels every special meaning of the symbols.
Example: ["? 0"] denotes the language {"? 0"}.

[A]

brackets are used to change the precedence of a regular relation.

(A)

parenthesis expresses optionality, and has the same meaning as $[A|0]$.

A B

Concatenation of the expressions or relations A and B.
Example: [[a b] [c d]] denotes the language {"ac", "ad", "bc", "bd"}

A ˆ n

Concatenation of A n times. Aˆ0 is defined as the empty string.
Example: [a]ˆ3 describes the language {"aaa"}.

A B

Union of the languages or relations A and B.
Example: [a b] describes the language {"a", "b"}.

A & B

Intersection of the languages A and B.
Example: [a b] & [a] describes the language {"a"}.

A - B

Minus of the languages A and B, and has the same meaning as [A & $\sim B$]
Example: [a b] - [a] describes the language {“b”}.

~A

Describes the complement of an expression, and has the same meaning as [?* - A]. Note that complement is always defined over an alphabet - the expression $\sim[A]$ is only unambiguous with respect to an alphabet.
Example: $\sim[a]$ denotes the language that doesn't contain the string “a”. If the alphabet is {a,b}, then $\sim[a]$ denotes the language {"", "b", "aa", "ba", ...}

A+

Repetition (Kleenes plus). A concatenated with itself an arbitrary number of times, including zero times.
Example: [a]+ denotes the infinite language {“a”,"aa","aaa",...}

A*

Kleene's star: [A+ 0].
Example: [a]* denotes the infinite language {"","a","aa",...}

$A

Containment. The set of strings where A appear at least once as a substring. Containment is the same thing as [?* A ?*]

A .x. B

Cross product of the languages A and B.
Example: [[a b] .x. c] describes the relations {("a", "c"), (“b", "c")}.

A .o. B

Composition of the relations A and B.
Example: [a:b c: d] .o. [d:e] describes the relation {("c","e")}.

The precedence of the operators is as follows, where 4 is the highest precedence:

4 ~ ˆ * + $

3 Concatenation

2 & -

1 .x. .o.

A file containing a program must end with *.fst, and an input file must end with *.dat. A program is a collection of functions defining regular relations. A function with zero arguments is called a **definition** or a **macro**.

A definition, or a macro, can for example look like this:

```
<digits> ::= "1" | "2" | "3" | "4" | "5" |
             "6" | "7" | "8" | "9" | "0";
```

and a function can look like this:

```
<swap,a,b> ::= b a ;
```

Note that strings are marked with quotes, and variables have no quotes.

Every program must contain a main definition (a program without a main definition will result in a parse error).

```
<main> ::= ... ;
```

The alphabet of a program is the symbols in the regular relation defined in the program.

1.4 Example program

The example was found at http://www.xrce.xerox.com/research/mltt/fst/fsexamples.html. The program simulates a soda machine and takes a collection of symbols that symbolises different coins. If the amount of money applied to the machine corresponds to 65 cent, then the machine produces a PLONK (enough money has been put into the machine to get a soda).

```
<nickel>  ::= ["n" .x. "c"^5];
<dime>    ::= ["d" .x. "c"^10];
<quarter> ::= ["q" .x. "c"^25];
<cent>    ::= ["c" .x. "c"];
<money>   ::= [ <nickel> | <dime> | <quarter> | <cent>]*;
<drink>   ::= ["c"^65 .x. "PLONK"];
<main>    ::= [ <money> .o. <drink> ];
```

1.5 Running FstStudio

fstStudio can be run in two modes, batch mode and interactive mode.

1.6 Batch mode

fst

Starts fstStudio in interactive mode. See below.

fst FILE [Options]

Run fstStudio in batch mode. FILE must end with *.fst, which defines a FstStudio program, or *.net, which defines a saved transducer. If no options is given, then input is taken from standard input, the transducer is applied down, and the output, if any, is produced on standard output.

Options	Meaning
-u	apply the transducer up
-d	apply the transducer up
-i FILE	take input from FILE
-o FILE	write output to FILE

1.7 Interactive mode

Interactive mode is entered by typing the following command:

```
$ fstStudio <return>

*****************************************************
* Welcome to Finite State Transducer Studio!        *
* Written purely in Haskell.                        *
* Version : 0.9                                     *
* Date    : 11 August 2001                          *
* Author  : Markus Forsberg                         *
* Please send bug reports/suggestions to:           *
* d97forma@dtek.chalmers.se                         *
*****************************************************

Type 'h' for help.

>
```

1.8 List of commands

r REG

Read a regular relation from standard input. If a regular expression is typed, then it is interpreted as the identity relation.

b

Build a epsilonfree, deterministic, minimal transducer from a loaded/typed regular relation.

bn

Build a epsilonfree, possibly non-deterministic, non-minimal transducer from a load/typed regular relation.

m

Minimize a built transducer.

det

Determinize a built transducer

s FILE

Save to FILE. If FILE ends with *.net, then the built transducer is saved. Any other suffix, saves the produced output in the system to FILE, if any.

l FILE

Load from FILE. FILE must end with *.fst, *.net or *.dat. If FILE ends with *.fst, then a FstStudio program is loaded into FstStudio. If FILE ends with *.net, then a transducer is loaded into FstStudio. If FILE ends with *.dat, then input is loaded into FstStudio.

l ab

Load and union two transducers. a and b must either be a file ending with *.net or the symbol *, which refers to the interior transducer. The produced transducer is possibly non-deterministic and non-minimal.

l a b

Load and concatenate two transducers. a and b must either be a file ending with *.net or the symbol *, which refers to the interior transducer. The produced transducer is possibly non-deterministic and non-minimal.

l a *

Load and apply Kleene's star on a transducer. a must either be a file ending with *.net or the symbol *, which refers to the interior transducer. The produced transducer is possibly non-deterministic and non-minimal.

l a .o. b

Load and compose two transducers. a and b must either be a file ending with *.net or the symbol *, which refers to the interior transducer. The produced transducer is possibly non-deterministic and non-minimal.

vt

View loaded/built transducer.

vr
: View loaded/typed regular relation.

vi
: View loaded input.

vo
: View produced output.

d
: Apply transducer down with loaded input.

u
: Apply transducer up with loaded input.

d SYMBOLS
: Apply tranducer down with SYMBOLS.

u SYMBOLS
: Apply transducer up with SYMBOLS.

c
: Clear memory.

h
: List commands.

q
: End session.

1.9 Example on FstStudio in interactive mode

First, save example program above to the file 'drink.fst'. Then start FstStudio in interactive mode.

```
>l drink.fst

Loaded a regular relation from drink.fst.

>b

Built a deterministic, minimal transducer with 26 states and 46 transitions.

>u PLONK

Input accepted. Type 'vo' to view outputs.
```

```
>vo

...
q n n d d d
q n n d d n n
q n n d n n d
q n n d n n n n
q n n d n d n
q d n q
q d n n d d
q d n n d n n
q d n n n n d
q d n n n n n n
q d n n n d n
q d n d n d
q d n d n n n
q d n d d n
q d q n
q d d d d
q d d d n n
q d d n n d
q d d n n n n
...

>s drink.out

Saved outputs to file drink.out.

>q

Do you really want to quit? (y):y

Session ended.
```

In 'drink.out' we now have all possible ways to feed the soda machine with coins, with regard to order, in such a way that we get a soda. There are 634 different ways.