Chapter 1

Programmer's Interface

Importing the module 'TransducerInterface.hs' gives you access to the following functions.

1.0.1 Regular expressions

Functions for constructing a simplified regular expression.

```
s          :: a -> Reg a              -- symbol
eps        :: Reg a                   -- epsilon
empty      :: Reg a                   -- empty set
allS       :: Reg a                   -- all symbol
star       :: Reg a -> Reg a          -- kleene's star
plus       :: Reg a -> Reg a          -- kleene's plus
complement :: Reg a -> Reg a          -- complement
(<|>)      :: Reg a -> Reg a -> Reg a -- union
(|>)       :: Reg a -> Reg a -> Reg a -- product
(<&>)      :: Reg a -> Reg a -> Reg a -- intersection
(<->)      :: Reg a -> Reg a -> Reg a -- set minus
symbols    :: Reg a -> a              -- collect all symbols.
```

1.0.2 Regular relations

Functions for constructing a simplified regular relation.

```
r          :: a -> a -> Reg a            -- relation
empty      :: RReg a                     -- empty set
idR        :: Reg a  -> RReg a           -- identity
star       :: RReg a -> RReg a           -- kleene's star
plus       :: RReg a -> RReg a           -- kleene's plus
(<|>)      :: RReg a -> RReg a -> RReg a -- union
(|>)       :: RReg a -> RReg a -> RReg a -- product
(<*>)      :: Reg a  -> Reg a  -> RReg a -- cross product
```

```
(<.>)      :: RReg a -> RReg a -> RReg a -- composition
symbols    :: RReg a -> a                -- collect all symbols.
```

1.0.3 Parsing regular relations

Functions for parsing regular relations.

'parseProgram' takes a string containing a fstStudio program, and try to parse it - if unsuccessful, it returns a error message. 'parseExp' parses a string containing a regular relation.

```
parseProgram :: String -> Either String (RReg String)
parseExp     :: String -> Either String (RReg String)
```

1.0.4 Construction and running

Functions for constructing and running a finite state transducer.

The function 'compile' construct a deterministic, epsilonfree, minimal transducer, and 'compileN' construct a epsilonfree, possibly non-deterministic, non-minimal transducer. The 'Sigma' type provides a way to add symbols that is not present in the regular relation. 'applyDown' and 'applyUp' are used to run the transducer.

```
type Sigma a = [a]

compile         :: Ord a => RReg a -> Sigma a -> Transducer a
compileN        :: Ord a => RReg a -> Sigma a -> Transducer a
determinize     :: Ord a => Transducer a -> Transducer a
minimize        :: Ord a => Transducer a -> Transducer a
unionT          :: Ord a => Transducer a -> Transducer a -> Transducer a
productT        :: Ord a => Transducer a -> Transducer a -> Transducer a
starT           :: Ord a => Transducer a -> Transducer a
compositionT    :: Ord a => Transducer a -> Transducer a -> Transducer a
emptyTransducer :: Transducer a
applyDown       :: Ord a => Transducer a -> [a] -> Maybe [[a]]
applyUp         :: Ord a => Transducer a -> [a] -> Maybe [[a]]
load            :: FilePath -> IO (Either String (Transducer String))
save            :: FilePath -> Transducer String -> IO (Either String ())
```

1.0.5 Transducer Information

Functions for getting information about a built transducer.

```
type State = Int
```

```
states              :: Transducer a -> [State]
isFinal             :: Transducer a -> State -> Bool
initial             :: Transducer a -> State
finals              :: Transducer a -> [State]
transitionsU        :: Transducer a -> (State,a) -> [(a,State)]
transitionsD        :: Transducer a -> (State,a) -> [(a,State)]
showTransducer      :: Transducer a -> String
numberOfStates      :: Transducer a -> Int
numberOfTransitions :: Transducer a -> Int
```