FLP – třetí cvičení
Simulátor Turingova stroje

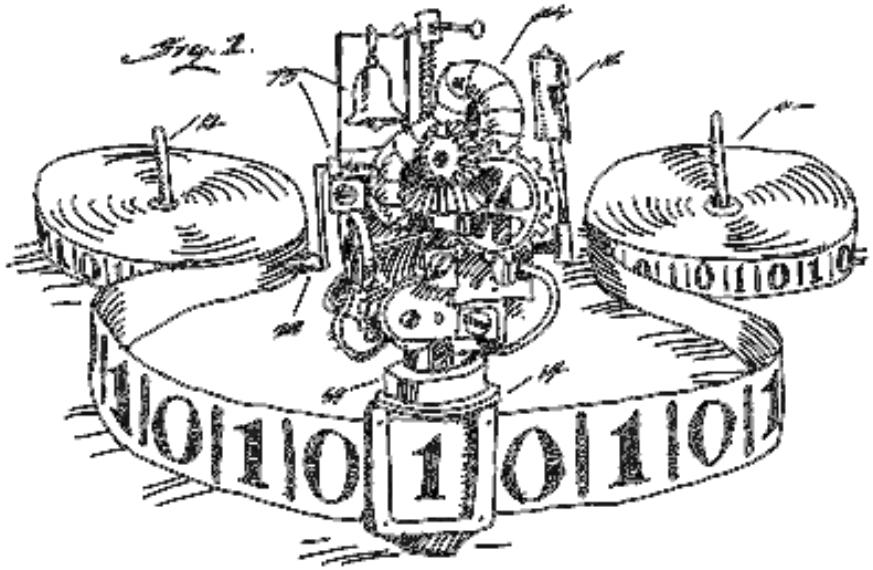
S. Židek, P. Matula, M. Kidoň, L. Škarvada

ÚIFS FIT VUT

Funkcionální a logické programování, 2019/2020

Zadání

Program, který simuluje výpočet Turingova stroje.



kresba: Duane Bibby

Turingův stroj

Turingův stroj je šestice $(Q, \Gamma, \Sigma, \delta, q_0, q_f)$

- Q je neprázdná konečná množina *stavů*
- Γ je neprázdná konečná množina *páskových symbolů*
- $\Delta \in \Gamma$ je prázdný páskový symbol
- $\Sigma \subseteq \Gamma - \{\Delta\}$ je množina *vstupních symbolů*
- δ je *přechodová funkce*: $\delta : (Q - \{q_f\}) \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$
- q_0 je *počáteční stav*
- q_f je *koncový stav*

Poznámka
Přechodová funkce δ může být parciální.

Páska a konfigurace TM

Páska Turingova stroje je nekonečná posloupnost $t_0, t_1, t_2, \ldots$ páskových symbolů.

Poznámka: Z další definice vyplývá, že TM bude pracovat pouze s páskou, na níž jsou *skoro všechny* symboly Δ.

Vstupní páska $w\,\Delta\,\Delta\,\Delta\,\Delta\ldots$, kde $w \in \Sigma^*$,
Turingova stroje se skládá ze vstupního slova následovaného prázdnými páskovými symboly.

Konfigurace Turingova stroje je uspořádaná trojice (q, t, p), kde $q \in Q$, t je páska a $p \geq 0$ je pozice hlavy na pásce. Počáteční konfigurace je konfigurace $(q_0, t_0, 0)$, kde q_0 je počáteční stav a t_0 je vstupní páska.

Výpočet TM

Výpočet Turingova stroje
je konečná posloupnost $(q_0, t_0, 0), \ldots, (q_n, t_n, p_n)$
nebo nekonečná posloupnost $(q_0, t_0, 0), (q_1, t_1, p_1), \ldots$
konfigurací, která začíná počáteční konfigurací a má následující vlastnosti (c_k je symbol na pozici p_k pásky t_k):
Pro všechna $0 \leq k < n$ je v konfiguraci $(q_{n+1}, t_{n+1}, p_{n+1})$

- $p_{k+1} = p_k - 1$ a $t_{k+1} = t_k$, pokud $\delta(q_k, c_k) = (q_{k+1}, \mathrm{L})$
- $p_{k+1} = p_k + 1$ a $t_{k+1} = t_k$, pokud $\delta(q_k, c_k) = (q_{k+1}, \mathrm{R})$
- $p_{k+1} = p_k$ a t_{k+1} má na pozici p_k znak c_{k+1}, pokud $\delta(q_k, c_k) = (q_{k+1}, c_{k+1})$

Je-li (q_n, t_n, p_n) poslední konfigurací v konečném výpočtu, pak $q_n = q_f$ nebo přechod $\delta(q_n, c_n)$ není definován.

Výsledek výpočtu $(q_0, t_0, 0), \ldots, (q_n, t_n, p_n)$ je obsah pásky t_n.

Specifikace

Podle voleb a parametrů příkazového řádku:

- `./turing-machine -s` *`soubor`*
  - načte popis TM ze souboru *soubor*
  - načte vstupní slovo ze standardního vstupu
  - simuluje výpočet TM na daném vstupu a v každém kroku vypisuje konfiguraci
- `./turing-machine -i` *`soubor`*
  - načte popis TM ze souboru *soubor* a uloží ho do vnitřní reprezentace
  - TM se převede zpět do textového tvaru a vypíše se na standardní výstup

Formát TM

1. seznam stavů oddělených čárkou
2. pásková abeceda (bez oddělovačů jednotlivých znaků)
3. počáteční stav
4. koncový stav
5. přechody; každý přechod je na zvláštním řádku a má jeden ze tří tvarů
   - q, c, q', c' je-li $\delta(q, c) = (q', c')$, $c' \in \Gamma$
   - $q, c, q', <$ je-li $\delta(q, c) = (q', \mathrm{L})$
   - $q, c, q', >$ je-li $\delta(q, c) = (q', \mathrm{R})$

Části programu

- TuringMain — hlavní program
  - přečtení příkazového řádku, nastavení podle `-i`/`-s`, provedení akce
  - výpis TM
  - výpis celého výpočtu
- TuringFuncs — funkce nad Turingovým strojem
  - nalezení následující konfigurace TM
  - krok výpočtu
  - výpočet
- TuringParse — načtení TM
  - textová analýza popisu TM (`Text.Parsec`)
  - validace
  - převod do vnitřní reprezentace
- TuringData — vnitřní reprezentace TM, datové typy
  - Turingův stroj
  - přechod a přechodová funkce
  - páska
  - konfigurace
  - stav, abeceda, akce

Program

```
module Main (main)  where

import System.Environment (getArgs)
import System.Exit (die)
import System.IO (readFile, getLine, putStr, putStrLn, hFlush, stdout)

import TuringData (TMachine(..), TMConfig(..), Tape(..))
import TuringFuncs (computation)
import TuringParse (parseTM)

main :: IO ()
main = do
    (action, input) <- procArgs =<< getArgs
    either die action (parseTM input)

-- Zpracování příkazového řádku
procArgs :: [String] -> IO (TMachine -> IO (), String)
procArgs [x,y] = do
    input <- readFile y
    case x of
     "-i" -> return (dumpTM, input)
     "-s" -> return (simulateTM, input)
     _    -> die ("unknown option " ++ x)
procArgs _ = die "expecting two arguments: [-i|-s] FILE"

-- Výpis na stdout při volbě '-i'
dumpTM :: TMachine -> IO ()
dumpTM tm = do
    putStrLn "dumping TM ..."
    putStr (show tm)

-- Načtení pásky a simulace TM při volbě '-s'
simulateTM :: TMachine -> IO ()
simulateTM tm = do
    putStr "input tape: " >> hFlush stdout
    inpLine <- getLine
    let headInp : tailInp = inpLine ++ repeat '_'
    putStrLn "simulating TM ..."
    -- Cv: DOPLŇTE SPRÁVNÉ ARGUMENTY FUNKCE computation MÍSTO undefined
    printComp (computation undefined undefined)
  where
    printComp comp = do
        mapM_ print (fst comp)
        putStrLn (snd comp)
```

```
{-# LANGUAGE RecordWildCards, TupleSections #-}

module TuringData where

import Control.Arrow (first)
import Control.Monad (join)
import Data.List (intercalate, dropWhileEnd, unfoldr)

type TState = String
type TSymbol = Char

-- Akce je posun nebo zápis symbolu
data Action = ALeft | ARight | AWrite TSymbol
  deriving (Eq)

instance Show Action where
    show ALeft = "<"
    show ARight = ">"
    show (AWrite c) = [c]

-- Pravidlo přechodové funkce
data Transition = Trans
    { fromState :: TState
    , fromSym :: TSymbol
    , toState :: TState
    , toAction :: Action
    } deriving (Eq)

instance Show Transition where
    show (Trans fq fs tq ta) = intercalate "," [fq, [fs], tq, show ta]

-- Celý Turingův stroj
data TMachine = TM
    { states :: [TState]
    , alphabet :: [TSymbol]
    , start :: TState
    , end :: TState
    , transRules :: [Transition]
    } deriving (Eq)

instance Show TMachine where
    show TM{..} = unlines $
        [intercalate "," states, alphabet, start, end] ++ map show transRules

-- Páska: symbol pod hlavou, symboly nalevo obráceně, symboly napravo
data Tape = Tape TSymbol [TSymbol] [TSymbol]

instance Show Tape where
    show (Tape x lts rts) =
        reverse (cut lts) ++ hili x ++ dropWhileEnd ('_' ==) (cut rts)
      where cut = take 30
```

```
-- Textová analýza Turingova stroje na vstupu
-- Využívá se knihovny Parsec

{-# LANGUAGE RecordWildCards #-}
module TuringParse (parseTM)  where

import Control.Applicative ((<$>), (<*>), (<$), (<*), (<|>))
import Control.Arrow (left)
import Control.Monad ((<=<))
import Data.Bool (bool)
import Text.Parsec (Parsec, parse,
        newline, alphaNum, string, char, satisfy, sepBy1, endBy, many1)
import Text.Parsec.String (Parser)

import TuringData

-- Převod TM z textu do vnitřní reprezentace
-- Funkce 'parse' aplikuje parser na řetězec a 'left show' převede chybový výsledek
-- na chybovou hlášku z typu 'Err TMachine'
parseTM :: String -> Err TMachine
parseTM = validate <=< left show . parse tmParser ""

-- Analýza celého TM
tmParser :: Parser TMachine
tmParser = TM <$> stateListP <* newline
              <*> alphabetP  <* newline
              <*> stateP     <* newline
              <*> stateP     <* newline
              <*> transitionsP
-- seznam stavů
stateListP :: Parser [TState]
stateListP = sepBy1 stateP comma

stateP :: Parser TState
stateP = many1 alphaNum

-- abeceda
alphabetP :: Parser [TSymbol]
alphabetP = many1 symbP

symbP :: Parser TSymbol  -- cokoliv, co není mezi vyjmenovanými znaky
symbP = satisfy (`notElem` " ,<>\n\t")

-- seznam pravidel, na každém řádku jedno
transitionsP :: Parser [Transition]
transitionsP = endBy transP newline

transP :: Parser Transition
transP = Trans <$> stateP <* comma <*> symbP <* comma <*> stateP <* comma <*> actP
  where actP  =  ALeft <$ char '<'  <|>  ARight <$ char '>'  <|>  AWrite <$> symbP

-- oddělovač
comma :: Parser Char
comma = char ','

-- Validační funkce: všechny stavy musí být v seznamu stavů a všechny symboly v abecedě
validate :: TMachine -> Err TMachine
validate tm@TM{..} = if allOK then Right tm else Left "invalid TM"
  where
    allOK = '_' `elem` alphabet
         && start `elem` states
         && end `elem` states
         && all ((`elem` states) . fromState) transRules
         && all ((`elem` alphabet) . fromSym) transRules
         && all ((`elem` states) . toState) transRules
         -- Cv: DOPLŇTE KONTROLU, ZDA PRAVIDLA PŘI ZÁPISU NA PÁSKU (AWrite) POUŽÍVAJÍ JEN SYMBOLY ABECEDY
         && undefined
```

```
{-# LANGUAGE RecordWildCards, TupleSections #-}

module TuringData where

import Control.Arrow (first)
import Control.Monad (join)
import Data.List (intercalate, dropWhileEnd, unfoldr)

type TState = String
type TSymbol = Char

-- Akce je posun nebo zápis symbolu
data Action = ALeft | ARight | AWrite TSymbol
  deriving (Eq)

instance Show Action where
    show ALeft = "<"
    show ARight = ">"
    show (AWrite c) = [c]

-- Pravidlo přechodové funkce
data Transition = Trans
    { fromState :: TState
    , fromSym :: TSymbol
    , toState :: TState
    , toAction :: Action
    } deriving (Eq)

instance Show Transition where
    show (Trans fq fs tq ta) = intercalate "," [fq, [fs], tq, show ta]

-- Celý Turingův stroj
data TMachine = TM
    { states :: [TState]
    , alphabet :: [TSymbol]
    , start :: TState
    , end :: TState
    , transRules :: [Transition]
    } deriving (Eq)

instance Show TMachine where
    show TM{..} = unlines $
        [intercalate "," states, alphabet, start, end] ++ map show transRules

-- Páska: symbol pod hlavou, symboly nalevo obráceně, symboly napravo
data Tape = Tape TSymbol [TSymbol] [TSymbol]

instance Show Tape where
    show (Tape x lts rts) =
        reverse (cut lts) ++ hili x ++ dropWhileEnd ('_' ==) (cut rts)
      where cut = take 30
            hili x = "[" ++ [x] ++ "]"

-- Konfigurace Turingova stroje
data TMConfig = TMConf TState Tape

instance Show TMConfig  where
    show (TMConf q tp) = "state " ++ q ++ "  tape: " ++ show tp

-- Typ výsledku nebo text případné chyby
type Err = Either String

-- Pomocná funkce; obdoba iterate, ale výsledek může být konečný seznam
-- Opakovaně aplikuje funkci na počáteční hodnotu
iterE :: (a -> Either b a) -> a -> ([a], b)
-- Cv: ZAPIŠTE DEFINICI JEDNÍM ŘÁDKEM POMOCÍ either
iterE g z = case g z of
    Left e  -> ([z], e)
    Right x -> first (z :) (iterE g x)
```

... viz soubory
TuringMain.hs,
TuringFuncs.hs,
TuringParse.hs,
TuringData.hs.

Náměty

Něco lze zavést a udělat alternativně

- Čtení vstupní pásky ze souboru
- Čtení vstupní pásky z příkazového řádku
- Možnost načíst stroj ze stdin
- Zobrazení konfigurací Turingova stroje

Něco lze vylepšit

- Zpracování příkazového řádku, `Options.Applicative`
- Leckde vhodnější datové struktury – např. přechodovou funkci reprezentovat nikoliv seznamem, ale množinou pravidel; rovněž stavy tvoří množinu

Něco udělat nelze

- *Který problém simulátorem TM nevyřešíme?*
  Lze napsat totální simulátor?

Požadavky a doporučení

- Citlivé a logické rozdělení programu do modulů.
- Vhodné datové typy odpovídající podstatě dat. Pryč jsou doby Lispu, kdy „všechno byl seznam“.
- Typové anotace.
- Totální funkce jsou bezpečnější než parciální.
- Vyhýbáme se hluboce zanořovaným konstrukcím.
- Neopakujeme stejné části kódu — DRY.
- „Nehaskellové“ hodnoty `unsafePerformIO`, `unsafeCoerce`, `unsafe...` nepoužíváme nikdy!
- Konsistentní odsazování, mezerami.
- Rozumně dlouhé řádky.
- Nástroj *hlint* pro kontrolu podezřelých i zbytečných výrazů.
- Nástroj *stack* nebo *Nix* pro vývoj haskellových projektů