1.

runParser: nimmt eine Parser Funktion und eine Liste an und wender die Funktion auf die Liste an.

symbol: Erwartet einen Character s mit dem man auf eine Liste vergleichen kann (ob der erste Character der Liste gleich s ist). Die Rückgabe, wenn dem der Fall ist, ist ein Tupel mit dem Rest der Liste zuerest und dem gleichen Character als zweites Paar. Wenn nicht, dann wird eine leere Liste zurückgegeben. Bsp: runParser (symbol 'a') ['a','b'] -> [("b",'a')]

token: Erwartet eine Liste von Characters und vergleicht ob diese ersten n Elemete gleich der ersten n Elemente der anderen Liste ist. Wenn nicht dann wird eine leere Liste zurückgegeben. Wenn ja, dann wird ein Tupel mit dem Rest als erstes und dem gleichen ersten n Elementen als zweites Paar zurückgegeben. Bsp: runParser (token ['b','a']) ['b','a'] -> [("","ba")]

satisfy: Die Funktion erwartet eine Funktion die checkt ob s True oder False ist. Ein Beispiel wäre die isDigit Funktion. Wenn die Liste leer ist, wird eine leere Liste zurückgegeben. Ansosten wird wieder ein Tupel mit der Restliste zuerst und als zweites Paar das Element welches True ist. Dies geschieht per List Comprehension und dem nachchecken ob x (das erste Element der Liste) angewendet auf die Funktion p (z.B isDigit) True ergibt. Wenn nicht wird wider eine leere Liste zurückgegeben. runParser (satisfy isDigit) ['1','c','b','a','c'] [("cbac",'1')]

succeed: Die Funktion gibt einen Tupel zurück, indem zuerst die Liste steht und als zweites das Element welches wir als Eingabe für v eingeben

fail: Gibt immer eine leere Liste aus

epsilon: ruft succeed auf mit der eingabe (). Also ist das zweite Element des Tupels ().

(<>*): verbindet zwei Parser (UND) symbol 'c'* <> symbol 'b' runParser (symbol 'c' <*> symbol 'b') "cb" [("", ('c','b'))]

(<|>): verbindert zwei Parser (OR), Da symbol c schon stimmt, wird b als rest ausgegeben runParser (symbol 'c' <|> symbol 'b') "cb" [("b",'c')]

(<!>): Zuerst werden 2 Parser verodert. Wenn die Liste nun leer ist, ist die Ausgabe eine leere Liste, wenn nicht dann ist die Ausgabe eine Liste von dem ersten Element der Liste runParser (symbol 'b' <!> symbol 'c') "c" [("",'c')]

sp: Erwartet einen Parser mit einem Char. Solange ein WhiteSpace in der Liste existert, werden Elemente aus der Liste gedroppt. Also werden Whitespaces am Anfang gedroppt. runParser (sp (token " abc")) "abcde" []

wsp: Filtert alle Characters die leer sind (' '). (Whitespaces) runParser (wsp (symbol ' ')) "b" []

just: runParser (succeed 'c') "b" [("b",'c')] ghci> runParser (just $ succeed 'c') "b" [] runParser (just (satisfy isDigit)) "1" [("",'1')]

(<@): Verlangt einen Parser und eine Funktion die a auf b anwendet. In der List comprehension wird der Parser auf die Liste angwendet und darauf wird dann die funktion angewendet runParser (symbol '1' <@ isDigit) "1" [("",True)]

some: Konvertiert einen Parser zu einem DetParser, indem das zweite Paar(z.B. bei symbol 'a' auf "a" steht im zweiten Parr a und im ersten "") mit dem Head aufgerufe wird (Der Tupel des Parser Typs ist in einer Liste, eine Liste von Tupeln) some (symbol 'a') "a" 'a'

(*<): verbindet einem Parser mit einem Parser wo <@ mit der Funktion fst aufgerufen wird. Also wird es mit dem was in fst tupel steht verbunden runParser (symbol 'c' < symbol 'v') ['c','v'] [("",'c')]*

(*>): Das gleiche nur mit snd runParser (symbol 'c' *> symbol 'v') ['c','v'] [("",'v')]

many: verknüpft parser p rekursiv mit anderem p der mit (<@) die Funktion liste verknüpft (Funktion anwendet, welche aus einem Tupel eine Liste macht). Dies wird dann mit (<!>) mit der Epsilon Funktion verknüpft runParser (many(symbol 'c' <*> symbol 'v')) ['c','v'] [("",[('c','v')])]

many1: gleiche wie many nur ohne den (<!>) Teil

option: Nun wird anstatt eine leere Liste entweder ein Nothing oder ein Just b übergeben runParser (option (symbol 'c')) "cabc" [("abc",Just 'c')] runParser (option (symbol 'c')) "abc" [("abc",Nothing)]

pack: Der Parser p wird mit den beiden anderen Parsern mit (*>) verknüpft, sodass wenn p in der mitte von den beiden Parsern s1 und s2 vorkommt, es p in snd schreibt runParser (pack (symbol 'd') (symbol 'c') (symbol 'v')) "dcvcabdcv" [("cabdcv",'c')]

ziffer: wenn eine Ziffer am Anfang steht, schreibt es diese in snd und den rest in fst, sonst leere liste runParser (ziffer) "1224" [("224",1)]

zahl: Funktion zahlToInt wird auf parser many1 ziffer ausgeführt, was dazu führt, dass mehr als nur die erste Zahl gecheckt wird runParser (zahl) "123" [("",123)]

chainl: liest und verbindet Strings mit einem weiteren String(Liste auf die chainl aufegerufen werden soll) (von links gelesen) runParser(chainl (zahl <@ show) (symbol '+' <@ (\c -> (\l r -> "(" ++ l ++ "+" ++ r ++ ")")))) "192+441+43" [("","((192+441)+43)")]

chainr: das gleiche wie chainl nur von rechts diesmal gelesen runParser(chainr (zahl <@ show) (symbol '+' <@ (\c -> (\l r -> "(" ++ l ++ "+" ++ r ++ ")")))) "192+441+43" [("","(192+(441+43))")]

integer: Wenn ein Minus vorkommt pack er das mit der darauffolgenden Zahl in fst. Er macht das, indem er die Funktion negate auf den Parser zahl mit <@ anwendet. runParser integer "-12-3" [("-3",-12)]

listOf, ap2, ap1, p <?@ und ziffernToInt fehlen -0,5 Punkte

2.

runParser (chainl zahl (symbol '+'> *succeed (+)* <|> *symbol* '-'> succeed (-))) "346 + 847 - 346 +223" [(" + 847 - 346 +223",346)] es wird von links gelesen. Der Parser Zahl liest die Zahlen. Der Parser symbol liest + und - gibt entweder die Funktion (-) oder (+) zurück ✓

runParser (chainl (chainl (chainl zahl (symbol ''> succeed *))) (symbol '/'*> succeed div)) (symbol'+'> *succeed (+)* <|> *symbol* '-'> succeed (-))) "12*34/2+56-789*" Hier wurde noch die Multiplikation und Division hinzufgefügt.

runParser (many ((option (symbol '+') > *integer)* <|> *(symbol* '-'> integer <@ negate))) "+984-234+445" Hier werden die Zahlen einzeln in eine Liste geschrieben. Wenn eine minus kommt, rufe die Funktion negate auf, welche Zahlen negiert. Ansonsten rufe den Parser integer auf.Wenn ein anderes Vorzeichen vorkommt (oder ein anderer Character als eine Zahl), dann schreibt er das in das fst anstatt dem snd.

4,5/6

Ergebnisse fehlen hier -1 Punkt