Domácí úkol 8: Zjednodušování aritmetických výrazů

V tomto domácím úkolu si vyzkoušíte práci se syntaktickými stromy v Haskellu. Vaším úkolem je naprogramovat několik funkcí pro práci s aritmetickými výrazy. Mějme následující datový typ pro reprezentaci jednoduchých aritmetických výrazů pomocí jejich syntaktického stromu:

```
data Expr a = Con a                  -- Constant value
            | Var String             -- Variable with name
            | Add (Expr a) (Expr a) -- Addition
            | Mul (Expr a) (Expr a) -- Multiplication
            deriving ( Show, Read, Eq )
```

Vaším úkolem je naprogramovat následující funkce:

- `eval :: (Num a) => Expr a -> [(String, a)] -> Maybe a`, která daný aritmetický výraz vyhodnotí pomocí dané valuace proměnných. Valuace proměnných je seznam dvojic ve tvaru `(název, hodnota)`, který přiřazuje proměnným ve výrazu hodnotu. V případě, že ve valuaci schází některá z potřebných proměnných, vyhodnotí se výraz na `Nothing`. Ve valuaci se žádná z proměnných nemůže vyskytnout vícekrát (tato situace se netestuje). Za tuto funkci můžete získat 0.2 bodu.
- `simplify01 :: Expr Integer -> Expr Integer`, která provede zjednodušení aritmetického výrazu. Tato simplifikace všechny podvýrazy ve tvaru `0 + x`, `x + 0`, `1 * x` nebo `x * 1` nahradí výrazem `x` a všechny podvýrazy ve tvaru `0 * x` nebo `x * 0` nahradí výrazem `0`, kde `x` je libovolný platný výraz. Tímto postupem je schopna výraz zjednodušit a snížit počet uzlů v jeho syntaktickém stromu. Za tuto funkci můžete získat 0.4 bodu.
- `simplifyConstants :: Expr Integer -> Expr Integer`, která identifikuje konstantní podvýrazy (tj. výrazy, ve kterých nefiguruje žádná proměnná), vyhodnotí je a nahradí je jedinou konstantou odpovídající jejich hodnotě. Za tuto funkci můžete získat 0.4 bodu.

Výsledné funkce by se měly chovat nějak takto:

```
> eval (Add (Con 42) (Mul (Var "a") (Var "b"))) [("a", 2), ("b", 3)]
Just 48
> eval (Add (Con 42) (Mul (Var "a") (Var "b"))) [("a", 2)]
Nothing

> simplify01 (Add (Con 42) (Mul (Var "a") (Var "b")))
Add (Con 42) (Mul (Var "a") (Var "b"))
> simplify01 (Add (Con 0) (Mul (Var "a") (Var "b")))
Mul (Var "a") (Var "b")
> simplify01 (Add (Con 0) (Mul (Con 1) (Var "b")))
Var "b"
> simplify01 (Mul (Var "d") (Mul (Con 4) (Con 0)))
Con 0

> simplifyConstants (Add (Con 0) (Mul (Var "a") (Var "b")))
Add (Con 0) (Mul (Var "a") (Var "b"))
> simplifyConstants (Add (Con 0) (Mul (Con 1) (Con 2)))
Con 2
> simplifyConstants (Add (Mul (Con 42) (Con 1)) (Mul (Var "a") ((Add (Con 4) (Con 1)))))
Add (Con 42) (Mul (Var "a") (Con 5))
```

Vaše řešení nemusí mít optimální časovou ani prostorovou složitost, ale vyvarujte se řešení, která mají exponenciální časovou složitost vzhledem k velikosti vstupního výrazu. Taková řešení nejenže nebudou hodnocena plným počtem bodů, ale také velice pravděpodobně neprojdou automatickými testy. Exponenciální řešení může vzniknout například v případě, ve kterém v definici funkce `f (Add x y)` použijete dvakrát podvýraz `f x`. V takovém případě nemáte zaručené, že se výraz `f x` vyhodnotí pouze jednou, a tedy celá funkce může mít exponenciální časovou složitost.

Při tvorbě vašeho řešení by se vám mohly hodit některé funkce z modulu `Data.List`. Můžete rovněž používat další moduly ze standardních balíků.[1]

Zvídavější studenty s hlubším zájmem o Haskell vyzýváme, ať si zkusí úlohu vyřešit pomocí funktorů a monád.[2] **Podotýkáme ale, že úloha je bez problémů řešitelná i pouze se znalostmi, které se přímo učí v tomto kurzu.**

Způsob odevzdání: domácí úkol se odevzdává přes odpovědník v ISu. Odpovědník obsahuje čtyři pole. Do prvního až třetího vložte vaši implementaci výše zmíněných funkcí, do posledního okomentujte, jak jste při řešení postupovali. Implementace každé funkce se vyhodnocuje nezávisle na ostatních (tzn. jako by každá funkce byla v samostatném souboru). Máte **dvě možnosti odevzdání celého odpovědníku** (a samozřejmě kontrolu syntaxe před odevzdáním). Dokud nekliknete na "odevzdat", můžete odpovědník zavřít a znovu otevřít bez penalizace. Po každém odevzdání uvidíte výsledky automatických testů, věnujte však čas dostatečnému vlastnímu testování. Následně budeme úkoly vyhodnocovat i ručně, bude se hodnotit i kvalita kódu a jeho popis, ne jen jeho funkčnost. Můžete získat až dva body, které se vám započítají do sumy bodů z domácích úkolů.

Bonus

K tomuto domácímu úkolu jsme se pro vás rozhodli nachystat i bonusovou část za další bod. Vaším úkolem je naprogramovat vlastní simplifikace aritmetických výrazů, které by měly co nejvíce snížit počet uzlů v syntaktickém stromu. Můžete naprogramovat libovolné simplifikace a různě je kombinovat – můžete např. propagovat konstanty, využít komutativity operátorů, můžete zkusit vytýkat apod. Musíte pouze zachovat ekvivalenci původního a vámi zjednodušeného výrazu.

Vaším úkolem je tedy implementovat funkci `simplify :: Expr Integer -> Expr Integer`, která provede zadané simplifikace. Připravili jsme pro vás soutěžní sadu 1000 výrazů, které má za úkol vaše funkce `simplify` co nejvíce zjednodušit. Tuto sadu najdete v souboru `DataSet.hs`, kde také naleznete funkci `evalSimplifications :: (Expr Integer -> Expr Integer) -> IO ()`, které předáte vaši simplifikační funkci a dozvíte se, o kolik uzlů se vám podařilo zmenšit testovací sadu.

Bonusový bod dostane 30 z vás, kteří dosáhnou největší nenulové redukce pomocí simplifikace, která je odlišná od simplifikací z povinné části úkolu, a zároveň je jejich funkce `simplify` korektní. Na vypracování bonusu jsou dva týdny času (detailně níže). Tento bod se nezapočítává do povinných 5 bodů z domácích úkolů nutných pro úspěšné absolvování předmětu – ovlivňuje pouze známku (podobně jako body ze cvičení – tento bod se ale nepočítá do limitu 5 bodů za aktivitu).

Způsob odevzdání: bonus vypracujte do samostatného souboru s názvem `simplify.hs`, který odevzdáte do odevzdávárny. Soubor musí být následujícího tvaru:

```
module Main ( main, simplify ) where

import DataSet

{- ruzne definice -}

simplify :: Expr Integer -> Expr Integer
simplify = {- telo funkce -}

main = evalSimplifications simplify
```

Protože soubor s daty se načítá opravdu dlouho, doporučujeme jej nejprve zkompilovat příkazem `ghc -c DataSet.hs` (respektive `ghc -c -dynamic DataSet.hs` pro GHC $\geq$ 7.10). Následně se pak při načítání řešení do interpretru použije zkompilovaná verze modulu `DataSet`.

Soubor s řešením by mělo být možné načíst do interpretru a odtud spustit vaši funkci `simplify` a rovněž jej musí jít zkompilovat a spustit (čímž se spustí evaluace na našich datech). Soubor v odevzdávárně můžete libovolně měnit do té doby, než bude uzavřena. Po jejím uzavření (týden po uzavření odpovědníku pro základní část) vaše řešení vyhodnotíme.

[1] tyto moduly najdete na https://hackage.haskell.org/package/base-4.7.0.0

[2] hezké povídání najdete na http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html