Domácí úkol 8: Formule predikátové logiky

Od devátého týdne se téma předmětu změní z funkcionálního programování na Prolog a logické programování. V něm hraje hlavní roli predikátová logika. Tento úkol je tak trochu překlenutím mezi oběma tématy: řešený bude v Haskellu, budete si v něm však hrát se syntaxí predikátové logiky prvního řádu.

Syntaxe predikátové logiky prvního řádu

Pro vyřešení tohoto úkolu nemusíte vědět nic o sémantice predikátové logiky (přívlastek „prvního řádu“ budeme od této chvíle vynechávat) mimo věcí známých z logiky výrokové (například význam logických spojek). Definujeme proto pouze syntaxi; Podrobnější informace si můžete nastudovat třeba na Wikipedii.

V predikátové logice pracujeme se dvěma typy objektů: s termy a formulemi.

Term je jakési zobecnění hodnoty. Konstanty jsou termy: např. a, 42, $brno$. Proměnné jsou termy: např. XS, N. Pozor, neplést s výrokovými proměnnými známými z výrokové logiky! Z konstant a proměnných je možné tvořit další hodnoty aplikováním funkcí: $\sin(45)$ i $f(g(brno), X)$ jsou termy.

Formálnější definice:

1. Proměnná je term.
2. Jsou-li t_1 až t_n (kde $n \in \mathbb{N}_0$) termy a f jméno n-ární funkce, pak $f(t_1, t_2, \ldots, t_n)$ je term. Při $n = 0$ se jedná o nulární funkci, což je vlastně konstanta, a zapisujeme ji bez závorek: f.
3. Nic jiného termem není.

Formule se podobají formulím výrokové logiky. Místo výrokových proměnných jsou ale jejich základním stavebním kamenem *predikáty*. Predikáty vyjadřují vlastnosti termů a relace mezi nimi. Jsou to vlastně funkce, jejichž argumenty jsou termy a výsledkem pravdivostní hodnota. Příkladem binárního predikátu je $\leq$. Ve formuli se může objevit třeba jako $\leq(0, N)$.

Podobně jako termy se formule definují induktivně:

1. Jsou-li t_1 až t_n (kde $n \in \mathbb{N}_0$) termy a P jméno n-árního predikátu, pak $P(t_1, t_2, \ldots, t_n)$ je formule. Při $n = 0$ se jedná o nulární predikát, což odpovídá výrokové proměnné, a zapisujeme jej bez závorek: P.
2. Jsou-li t_1 a t_2 termy, potom jejich porovnání $t_1 = t_2$ je formule.
3. Je-li φ formule, je $\neg\varphi$ formule.
4. Jsou-li φ a ψ formule, je $(\varphi \wedge \psi)$ formule.
5. Jsou-li φ a ψ formule, je $(\varphi \vee \psi)$ formule.
6. Jsou-li φ a ψ formule, je $(\varphi \Rightarrow \psi)$ formule.
7. Je-li φ formule a X proměnná, je $\exists X\, \varphi$ formule.
8. Je-li φ formule a X proměnná, je $\forall X\, \varphi$ formule.
9. Nic jiného formulí není.

Zadání

Stáhněte si kostru řešení. V ní už jsou definovány datové typy reprezentující termy a formule predikátové logiky: `Term` a `Formula`. Veškerá jména funkcí, proměnných i predikátů jsou typu `String` a neklademe na ně žádná omezení. Argumenty funkcí a predikátů jsou umístěny do seznamu, kde první prvek odpovídá prvnímu argumentu atd. Arita funkcí a predikátů je tak přesně dána délkou seznamu argumentů. Funkce a predikáty se stejným názvem se mohou vyskytovat v různých aritách.

Úloha má tři části. První je funkce pro převod formule do NNF, neboli *negation normal form* (Wikipedie[en]). Druhou je funkce pro ověření, že kvantifikátory jsou ve formuli použity smysluplně. Třetí podúlohou je funkce pro hezké a čitelné zobrazení formulí a termů. Tyto části jsou na sobě nezávislé, ale pro snazší ladění prvních dvou může přijít vhod mít už implementované vypisování.

1. Negation normal form (NNF)

Napište funkci `nnf :: Formula -> Formula`, která zadanou formuli převede do NNF. Aby byla formule v NNF, musí být splněny dvě podmínky:

1. Negace se v ní nesmí vyskytovat jinde než těsně u predikátu či porovnání.

   Formálněji: V každé podformuli tvaru $\neg\varphi$ je φ buď tvaru $P(t_1, \ldots, t_n)$, nebo tvaru $t_1 = t_2$.

2. Nevyskytuje se v ní spojka implikace („$\Rightarrow$“).

Na funkci `nnf` navíc klademe jedno omezení technického rázu: predikáty a porovnání musí být ve výsledku ve stejném pořadí jako na vstupu. Jinými slovy neprohazujte strany operátorů a myslete na to i při přepisování implikace.

Postup převodu do NNF je popsán třeba na anglické Wikipedii, ale vystačíte si s De Morganovými zákony pro výrokovou logiku a následujícími ekvivalencemi:

- $\neg\neg\varphi \equiv \varphi$
- $\neg\forall X\varphi \equiv \exists X\neg\varphi$
- $\neg\exists X\varphi \equiv \forall X\neg\varphi$

Vyhodnocení příkladových formulí z kostry:

```
nnf formula2 ⇝*
And (Equal (Var "A") (Fun "f" [Var "A"]))
    (Not (Equal (Var "A") (Fun "g" [Fun "c" []])))
nnf formula4 ⇝*
Forall "x" (Or (Not (Pred "P" [Var "x"]))
               (Or (Not (Pred "P" []))
                   (Pred "P" [Fun "x" [],Fun "x" [Var "x"]])))
nnf formula5 ⇝*
Exists "A" (Forall "B" (And (Or (Not (Pred "P" [Var "A",Fun "5555" []]))
                                (Not (Pred "Q" [Fun "5555" [],Var "B"])))
                       (Or (Not (Pred "R" [])) (Not (Pred "S" [Var "B"])))))
```

2. Smysluplnost formule

Napište funkci `validate :: Formula -> Bool`, která o zadané formuli rozhodne, zda je smysluplná. Za smysluplnou označujeme takovou formuli, která splňuje následující dvě podmínky:

1. Ve formuli se nevyskytují nekvantifikované proměnné (tj. je *uzavřená*).

   Kvantifikovanost: Proměnná X je ve formuli φ kvantifikovaná, pokud se vyskytuje pouze v jejích podformulích tvaru $\forall X\psi$ nebo $\exists X\psi$.

2. Tatáž proměnná není kvantifikována vícekrát.

   Formálněji: Mějme podformuli tvaru $\exists X\varphi$ nebo $\forall X\varphi$, kde X je zvoleno libovolně. Potom φ neobsahuje podformuli tvaru $\exists X\psi$ nebo $\forall X\psi$.

Všimněte si, že se i ve smysluplné formuli mohou vyskytovat kvantifikované proměnné, které nejsou použity v žádném termu. Také si povšimněte, že ve smysluplné formuli mohou být kvantifikovány dvě proměnné se stejným názvem, pokud se nachází v disjunktních podformulích (a jsou to tedy různé proměnné).

Vyhodnocení příkladových formulí z kostry:

```
validate formula1 ⇝* True
validate formula2 ⇝* False
validate formula3 ⇝* False
validate formula4 ⇝* True
validate formula5 ⇝* True
```

3. Pěkný výpis

Definujeme typovou třídu `Pretty`, která předepisuje jednu funkci `pprint :: Pretty a => a -> String`. Funkce `pprint` má termy a formule vypisovat tak, jak bychom je napsali v matematice. A protože jsme v roce 2018, budeme ve výpise používat i jiné znaky než ty patřící do půl století starého ASCII.

3.1. Pěkné termy

Instance typu `Term` pro `Pretty` bude poskytovat následující zobrazení termů:

1. Proměnné jsou reprezentovány pouze svým jménem.
2. Nulární funkce jsou reprezentovány pouze svým jménem.
3. Aplikace funkce na termy je reprezentována jménem funkce následovaným v kulatých závorkách uzavřeným seznamem argumentů oddělených čárkou.

Na počtu mezer při automatických testech nezáleží, ale snažte se o rozumně vypadající výstup.

Příklady:

```
pprint (Var "X") ⇝* "X"
pprint (Fun "42" []) ⇝* "42"
pprint (Fun "f" [Fun "g" [Var "X"]]) ⇝* "f( g( X ) )"
pprint (Fun "exec" [Var "Ord", Fun "66" []]) ⇝* "exec( Ord, 66 )"
```

3.2. Pěkné formule

Instance typu `Formula` pro `Pretty` bude poskytovat následující zobrazení formulí:

1. Predikáty se vypisují podobně jako funkce:
   a. nulární predikáty jsou reprezentovány pouze svým jménem,
   b. ostatní jsou reprezentovány jménem predikátu následovaným v kulatých závorkách uzavřeným seznamem argumentů oddělených čárkou.
2. Porovnání termů `Equal t1 t2` je vypsáno jako `t1 = t2` (bez závorek kolem)
3. Negace formule `Not fml` je vypsána jako `¬fml`.
4. Konjunkce a disjunkce formulí jsou vypsány s infixově zapsaným příslušným operátorem a s kulatými závorkami kolem:
   a. `And f1 f2` se vypíše jako `(f1 ∧ f2)`,
   b. `Or f1 f2` se vypíše jako `(f1 ∨ f2)`.

   Ale pozor: Výpis navíc bere v potaz asociativitu obou operací a vnořené konjunkce zobrazí jako jedinou závorku s více konjunkty. Totéž pro disjunkce.
   a. Například `((a ∧ b) ∧ (c ∧ d))` ani `(((a ∧ b) ∧ c) ∧ d)` není přípustné. Správné zobrazení je `(a ∧ b ∧ c ∧ d)`.
   b. Například `((a ∧ (b ∨ c)) ∧ d)` má být zobrazeno jako `(a ∧ (b ∨ c) ∧ d)`, rozhodně ne však jako `(a ∧ d ∧ (b ∨ c))`, neboť komutativity se nevyužívá.
5. Implikace `Implies x y` je vypsána jako `(x ⇒ y)`. Žádné závorky se při vícenásobných implikacích nevynechávají.
6. Existenční kvantifikace `Exists "X" fml` je vypsána jako `∃X fml`.
7. Univerzální kvantifikace `Forall "X" fml` je vypsána jako `∀X fml`.

Veškeré podformule a termy se samozřejmě opět vypisují *pěkně*, tedy funkcí `pprint`. Na počtu mezer kolem operátorů, závorek apod. automatickým testům nezáleží. Všechny potřebné symboly mimo rozsah ASCII jsou umístěny i v kostře řešení, můžete je proto kopírovat na patřičná místa v kódu.[1]

Vyhodnocení příkladových formulí z kostry:

```
pprint formula1 ⇝* "∃z ∀x ∃y x = y"
pprint formula2 ⇝* "¬(A = f(A) ⇒ A = g(c))"
pprint formula3 ⇝* "(P(3) ∧ Q(X) ∧ X = 3)"
pprint formula4 ⇝* "∀x (P(x) ⇒ (P ⇒ P(x, x(x))))"
pprint formula5 ⇝* "¬∀A ∃B ¬((P(A, 5555) ⇒ ¬Q(5555, B)) ∧ (¬R ∨ ¬S(B)))"
```

[1] Je ale možné, že váš operační systém nabízí přímější metodu přes kód znaku. Používáte-li linuxovou distribuci s grafickým rozhraním, můžete si nastavit klávesu Compose. Uživatelé textového editoru *Vim* mohou s výhodou využít digrafy.

Zadáte-li však tyto příklady do interpretu, jím použitá funkce `show` nahradí znaky mimo rozsah ASCII zpětným lomítkem následovaným číselným kódem znaku. Vyřešit to můžete použitím funkce `putStrLn` nebo obalením řetězců do vlastního datového typu a napsáním jeho neescapující instance pro `Show`. Je to jednodušší, než by se mohlo na první pohled zdát.

Poznámky a tipy

- Směle využívejte funkcí z `Prelude`.
- Importovat smíte i jiné moduly z balíku `base`, prohlédněte si kupříkladu `Data.List`.
- Nepište podobné funkce pořád dokola, ale pokuste se problém co nejvíc abstrahovat.
- Věnujte chvilku přemýšlením o efektivitě vašeho řešení. Procházet formule příliš mnohokrát, dá-li se problém vyřešit jedním průchodem, není hezké. **Pozor:** pokud bude vaše řešení příliš neefektivní, může při automatickém testování vypršet časový limit a vy nezískáte žádné body.
- Přebíráte-li kód odjinud, nezapomeňte uvést zdroj. V opačném případě bude na vaši práci pohlíženo jako na plagiát.
- Nezapomeňte, že **opisování** je **zakázáno** a bude postihováno podle studijního řádu.
- V diskusním fóru si přečtěte, čeho se v kódu vyvarovat, aby byli opravující spokojeni s úhledností vašeho řešení.

Odevzdání a bodování

Domácí úkol se odevzdává přes odpovědník v ISu. Tento odpovědník obsahuje jediné pole, do kterého vložíte svou implementaci požadovaných funkcí, včetně potřebných importů, avšak bez definic z kostry. Tato úloha opět podléhá ruční kontrole cvičícími, při níž se přihlíží i k jasnosti a pochopitelnosti řešení. Myšlenku svého řešení proto **stručně** komentujte v kódu. Je-li kód srozumitelný sám o sobě, nemusí být žádný komentář zapotřebí.

Máte **dvě možnosti odevzdání** (a samozřejmě kontrolu syntaxe před odevzdáním). Dokud nekliknete na „odevzdat“, můžete odpovědník zavřít a znovu otevřít bez penalizace. Nezapomínejte na průběžné ukládání. Po každém odevzdání uvidíte výsledky automatických testů, věnujte však dostatek času vlastnímu testování.

Pokud vaše řešení projde všemi automatickými testy, obdržíte jeden bod. V případě, že se vám podařilo implementovat jen část funkcí, sice žádný bod automaticky nedostanete, ale cvičící vám podle míry vyřešenosti při ruční kontrole úlohy udělí část bodu. Za kvalitu kódu a jeho efektivitu vám cvičící mohou udělit další jeden bod nebo jeho část.

Pozor: cvičící budou vždy opravovat vaše poslední odevzdání.

Bonus

V tomto úkolu můžete navíc získat ještě jeden bod navíc, doplníte-li řešení o funkcionalitu netriviálně používající vstup a výstup (a tedy typový konstruktor `IO`).[2]

Například můžete napsat funkci typu `Formula -> IO ()`, která pro zadanou formuli spustí interaktivní znázornění převodu do NNF (tedy jakýsi step-by-step výukový režim). Mělo by být možné minimálně ručně odkrokovat postup převodu, ale klidně se můžete rozvášnit a implementovat i krokování dozadu či volitelný dodatečný popis toho, jaká úprava se zrovna provádí.

Navrženým zadáním se ale vůbec nemusíte řídit, váš bonus by jen neměl opouštět téma predikátové logiky a, jak již bylo řečeno, musí netriviálním způsobem používat `IO`.

Bonusovou část odevzdávejte jako samostatný soubor do odevzdávárny v ISu. Soubor musí být přeložitelný sám o sobě; musí tedy obsahovat definice datových typů a typové třídy ze zadání. Na začátku souboru stručně popište implementovanou bonusovou funkcionalitu a způsob použití. Na bonusovou část máte více času – odevzdávárna se uzavře až na konci **neděle 25. listopadu**.

[2] Chcete-li se při řešení bonusu ještě i naučit něco nad rámec základního kurzu, můžete využít i jiných monád než `IO`. Rozhodnete-li se držet navrženého bonusu, můžete elegantně využít toho, že dvojice se chová jako monáda písaře.