IB015 – Domácí úkol 10: Výrokové formule

Druhý velký úkol se bude celý točit kolem formulí výrokové logiky. Ty si budeme representovat následující dvojicí typů:

```
data PropFormula = Const Bool     -- píšeme tučným T a F
                 | Var String
                 | Bin BinOp PropFormula PropFormula
                 | Not PropFormula
                deriving (Show, Eq)

data BinOp = And | Or | Implies deriving (Show, Eq)
```

Jejich význam by měl být zřejmý a ve zbytku zadání se budeme bavit přímo o formulích, nikoli jejich representaci hodnotami typu `PropFormula`. Typy v kostře neměňte.

Doporučení na začátek

Řešení úlohy vám mohou usnadnit (a zpřehlednit) dva syntaktické prvky jazyka, které jste doteď možná vůbec nepoužívali. V přednášce či na internetu hledejte:

- **stráže** (*guards*), které se umožní vyhnout `if`ům a podmínky psát na stejnou úroveň jako vzory, a
- **zavináč**, který umožní pracovat se vzorem i jeho částmi: např. `f@(Bin _ l r)` naváže na `f` celý výraz a na `l` a `r` jeho podvýrazy.

Rozehřátí

Nejprve pár lehoučkých funkcí, které se vám ale budou hodit později.

▷ `isLit :: PropFormula -> Bool`
Rozhodněte, zda je formule literálem. *Literál* je proměnná či její negace.

▷ `isConst :: PropFormula -> Bool`
Rozhodněte, zda se jedná o konstantu.

Prostřihováváme košaté formule

Vaším dalším úkolem bude naprogramovat dvě strategie zmenšování formulí bez změny jejich významu. Obě mají velice podobný princip, odolejte tak pokušení kopírovat kód. Namísto toho zkuste obě napsat s využitím stejné pomocné funkce.

1. `evalConsts :: PropFormula -> PropFormula`
   Strategie spočívá ve vyhodnocení všech podvýrazů neobsahujících proměnné a nahrazení těchto podvýrazů rovnou výsledkem jejich vyhodnocení.
2. `removeConsts :: PropFormula -> PropFormula`
   Strategie odstraní všechny vnitřní konstanty nahrazujíc každou operaci, která má konstantní alespoň jeden operand, jednodušším výrazem. Konstanta se tedy na výstupu může objevit jen tehdy, smrskne-li se celá formule na konstantu.

Například formule

$$(\mathbf{F} \vee (A \wedge \mathbf{T})) \wedge (\neg A \wedge (\mathbf{T} \vee (\mathbf{F} \wedge \mathbf{T})))$$

se první strategií zjednoduší na

$$(\mathbf{F} \vee (A \wedge \mathbf{T})) \wedge (\neg A \wedge \mathbf{T})$$

a druhou dokonce až na

$$A \wedge \neg A$$

Na oba způsoby zjednodušování stačí jeden průchod výrazem. Speciálně na to je vyčleněna část bodů udělovaných cvičícím. Tu v žádném případě nezískáte za řešení, které na výraz aplikuje funkci pro jeden krok zjednodušení tak dlouho, dokud se výraz zmenšuje.

Jistě byste sami přišli na další způsoby zmenšování, při nichž formule zůstanou logicky ekvivalentní. Nedělejte to. Testy očekávají pro každý vstup jeden konkrétní výstup, a to takový, který vznikne pouze popsanými zmenšovacími strategiemi.

Loučíme se se spojkami

Zbavivše se konstant se nyní vrhneme trochu uklidit v operátorech. Doposud jsme uvažovali systém čtyř spojek: $(\wedge, \vee, \to, \neg)$. Ten ale víme, že není minimální – můžeme jej zmenšit na $(\wedge, \vee, \neg)$ nebo na $(\to, \neg)$ a přitom umět vyjádřit totéž. My ve druhém případě navíc vyměníme negaci za konstantu *nepravda* a zaměříme se na $(\to, \mathbf{F})$.

Vaším druhým úkolem bude převádět libovolné formule *bez konstant* na tvar používající pouze tyto systémy spojek.

1. `toNNF :: PropFormula -> PropFormula`
   Převeďte formuli do systému $(\wedge, \vee, \neg)$. Navíc budete mít vedlejší úkol: zbavit se všech negací s výjimkou těch těsně u proměnných (jinými slovy „protlačit“ negace k proměnným). Vlastně tak chceme formuli převést do negační normální formy. Použijte k tomu De Morganova pravidla (ve druhé přednášce IB000 jsou uvedena jako „převod na normální formu“).

   Jako technické omezení požadujeme, aby převod příliš neměnil strukturu výrazu: nedějte tedy žádné chytré optimalisace, neprohazujte operandy a implikaci přepisujte striktně jako

   $$\varphi \to \psi \quad \Rightarrow \quad \neg\varphi \vee \psi$$

   S De Morganovými pravidly budete v suchu. Příklad převodu:

   $$(\neg A \vee B) \to \neg(B \wedge \neg C) \quad \Rightarrow \quad (A \wedge \neg B) \vee (\neg B \vee C)$$

2. `toIF :: PropFormula -> PropFormula`
   Pro převod do implikační formy $(\to, \mathbf{F})$ použijte pouze následující převodní pravidla:

   $$\begin{aligned} \varphi \wedge \psi &\Rightarrow \neg(\varphi \to \neg\psi) \\ \varphi \vee \psi &\Rightarrow \neg\varphi \to \psi \\ \neg\varphi &\Rightarrow \varphi \to \mathbf{F} \end{aligned}$$

   Příklad převodu:

   $$A \wedge \neg(B \to \neg C) \quad \Rightarrow \quad (A \to (((B \to (C \to \mathbf{F})) \to \mathbf{F}) \to \mathbf{F})) \to \mathbf{F}$$

Oba převody opět vyřešte na jeden průchod výrazem, jinak nemůžete získat všechny body přidělované člověkem.

Formy na kanóny

Počítače a teoretičtí informatici rádi pracují s formulemi v kanonických formách. My se zaměříme na tyto dvě:

1. konjunktivní normální forma (KNF), která má tvar konjunkce disjunkcí literálů, a
2. disjunktivní normální forma (DNF), která má tvar disjunkce konjunkcí literálů.

Konjunkce i disjunkce mohou být i jednoprvkové: samotný literál nebo celý výraz. Například tedy:

- A, $A \wedge \neg B$ či $\neg A \vee B \vee \neg C$ jsou v KNF i DNF;
- $(A \vee B) \wedge (A \vee \neg C \vee \neg D) \wedge \neg E$ je jen v KNF a
- $(A \wedge \neg B \wedge C) \vee B \vee (\neg C \wedge D)$ je jen v DNF.

Všimněte si, že některé z uvedených formulí mají více representací lišících se tím, kam umístíme vynechané závorky v konjunkcích či disjunkcích více než dvou podvýrazů.

Konstanty se v těchto kanonických formách vůbec nevyskytují[1], takže například není možné vyjádřit nepravdu bez použití proměnných (jde ale použít například $A \wedge \neg A$).

▷ `isCNF :: PropFormula -> Bool`
▷ `isDNF :: PropFormula -> Bool`
Nebudeme po vás chtít naprogramovat převod formulí do těchto forem, ale pro každou z nich jen funkci rozhodující, zda je zadaná formule v oné kanonické formě.

[1] byť v jistém smyslu je **T** konjunkce a **F** disjunkce nula prvků, stejně jako 1 je prázdný součin a 0 prázdný součet.

Obě funkce mají velmi podobnou strukturu; vyhněte se ale kopírování kódu. Místo toho zkuste navrhnout jejich vhodné zobecnění. Vzpomeňte si také, že jste si už napsali funkci rozhodující, zda je formule literálem. Vyhnutí se duplikaci bude opět oceněno ručně přidělovanými body.

▷ `renameVars :: PropFormula -> PropFormula`
Jednu kanonizaci po vás ale přeci chtít budeme – přejmenování proměnných ve formuli na velká písmena anglické abecedy, a to postupně zleva od A. Stejné proměnné na vstupu musí přirozeně zůstat stejnými i ve výsledku. Například

$$\neg(Hask \wedge Ell) \to (\neg Hask \vee (Pop \wedge Ell))$$

se takto kanonizuje na

$$\neg(A \wedge B) \to (\neg A \vee (C \wedge B))$$

Můžete předpokládat, že různých proměnných nebude více než 26. Bude se vám hodit knihovní funkce `lookup :: Eq k => k -> [(k, v)] -> Maybe v`. Jistě využijete i vzory v lokální definici:

```
let (a, b) = funkceVracejiciDvojici x y z in a + b.
```

Testy

V kostře máte několik základních testů. Spustíte je vyhodnocením funkce `main` v interpretu nebo příkazem `runghc 10post4.hs` v terminálu. Je jich ale velice málo; hlavním účelem je vůbec vám umožnit testovat své řešení pohodlně. Přidávejte proto vlastní testovací případy – je to lepší než dokola vyhodnocovat tytéž vstupy v interpretu a kontrolovat ručně výsledky. Stačí do seznamu testů nějaké funkce přidat dvojici `(vstup, výstup)` a `main` pak bude vždy ověřovat, zda vaše funkce na `vstupu` vrací `výstup`.

Během ručního testování můžete taky použít dodanou funkci `pprint`, která vypisuje formule v příjemně čitelné podobě s unikódovými symboly.

Odevzdání a bodování

Technické detaily k odevzdávání jsou stejné jako v minulém velkém úkolu (8. po 3. cvičení), proto si je připomeňte tam. Ve zkratce – vypracované řešení v jednom textovém souboru s koncovkou `.hs` vkládejte do správné odevzdávárny v Informačním systému; nezapomeňte otypovat všechny funkce.

Situace s bodováním je tentokrát trochu složitější, protože body chceme rozdávat i za věci, které se nedají automaticky testovat. Chcete-li taktizovat, můžete využít následující přehled rozdávaných bodů:

Funkce	Body	Uděluje
`isLit` ∧ `isConst`	0,1	stroj
`evalConsts`	0,2	stroj
… na jeden průchod	0,1	člověk
`removeConsts`	0,3	stroj
… na jeden průchod	0,2	člověk
`toNNF`	0,2	stroj
… na jeden průchod	0,2	člověk
`toIF`	0,2	stroj
… na jeden průchod	0,1	člověk
`isCNF` ∨ `isDNF`	0,2	stroj
… obě, bez duplikace	0,2	člověk
`renameVars`	0,5	stroj
Kvalita kódu	0,5	člověk

Pro jistotu připomínáme, že **spolupráce více studentů je zakázaná**, kontrolujeme ji a postihujeme ji mj. stržením všech bodů za úlohu.

S jakýmikoli dotazy se obracejte do diskusního fóra. **Pozor**, naše příspěvky na fóru budou plnohodnotným doplněním zadání a ne všechno se dostane přímo do tohoto souboru, pravidelně proto úlohové vlákno čtěte. Příjemné programování!