FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z předmětu

Úvod do znalostního inženýrství

Splnitelnost logického výrazu

17. prosince 2023

Dominik Vladař
A21B0317P
vladar21@students.zcu.cz

Obsah

1 Zadání

Je dán logický výraz se spojkami `and`, `or`, `not`, `xor`, `impl`, `equiv` (definujte pomocí operátorů) obsahující logické konstanty `true`, `false` a proměnné (reprezentované atomy). Vytvořte program, který rozhodne, zda je zadaný výraz pravdivý, resp. splnitelný, případně vydá nutné ohodnocení proměnných tak, aby byl výraz splněn. Např.: `?-sat((x or not x) and y, R).` vrací `R=[y/true]`.

2 Analýza úlohy

Před implementací úlohy bylo nutné vybrat řešení, které bude dostatečně efektivní. Jednodušším řešením, které bylo při výběru algoritmu zvažováno, bylo použití metody **brute force**, pomocí které by byly postupně za všechny proměnné dosazeny do výrazu hodnoty *true* a $false$, a poté bychom výraz vyhodnotili postupným aplikováním přepisovacích pravidel typu `if false and true then false`. Pokud je výsledkem vyhodnocení *true*, znamená to, že pro konkrétní ohodnocení proměnných je výraz splnitelný. Tento algoritmus je na pro náš problém použitelný, ale vzhledem k tomu, že jeho složitost je exponenciální vzhledem k počtu proměnných, je vhodné hledat řešení, pomocí kterého se nám podaří tuto výpočetní složitost eliminovat.

V programu je použité jiné řešení, které nejdříve výraz co nejvíce zjednoduší pomocí přepisovacích pravidel zmíněných výše (např. výraz `(x and (true or false)) or y` bude zjednodušen na výraz `x or y`), poté nahradí první proměnnou za *true* a $false$ (dostaneme tak dvojici výrazů `true or y` a `false or y`), tyto výrazy opět maximálně zjednoduší (dostaneme `true` a `y`) a postupuje rekurzivně dál, takže najde další doposud nezjednodušenou proměnnou (v našem případě y) a nahradí ji za *true*, $false$. Stejně jako u metody brute force, pokud je výsledkem zjednodušení *true*, znamená to, že pro konkrétní ohodnocení proměnných (některé proměnné mohly během zjednodušení vypadnout, to by znamenalo že výsledek na nich není závislý a mohou nabývat hodnot *true* i $false$) je výraz splnitelný. Pokud použijeme toto řešení, můžeme navíc používat přepisovací pravidla s proměnnými, například `if promenna and true then promenna`. Toto řešení bylo vybráno z důvodu snížení výpočetní náročnosti programu, ačkoliv se najdou (zejména jednodušší) logické výrazy, u kterých je postupné vyhodnocování zbytečnou přítěží. U složitějších výrazů má ale toto řešení naopak velký potenciál snížit dobu běhu programu. Detailnější popis vybrané metody řešení problému je zmíněn v

kapitole Popis algoritmu.

3 Popis algoritmu

Program je koncipován jako pravidlový znalostní systém. Pro vyhodnocení, za jakých podmínek je logický výraz pravdivý nejdříve převede zadaný výraz do lépe zpracovávatelné podoby seznamu, např. `(x or (true and y)) or z` bude vyjádřeno jako `[[x, or, [true, and, y]], or, z]`, kde každý prvek seznamu je ve smyslu jazyka Python objektem, se kterým je možné nadále pracovat. Při tomto přidávání prvků do seznamů jsou také zaznamenávány jedinečné názvy proměnných.

Poté je výraz zjednodušován pomocí přepisovacích pravidel. Na výraz se postupně zkouší aplikovat všechna definovaná přepisovací pravidla, a pokud alespoň jedno uspěje a podaří se díky tomu výraz zjednodušit, zjednodušování bude pokračovat dalším aplikováním všech pravidel. Pokud ale ani jedno přepisovací pravidlo nebude možné aplikovat, zjednodušování končí a výraz již není možné více zjednodušit. Pokud díky aplikování nějakého pravidla zbyde v závorkách (reprezentovány seznamem) poslední prvek, závorky jsou odstraněny (jediný prvek seznamu nahradí původní seznam). Když algoritmus při svém běhu narazí na neočekávaný prvek výrazu (např. jediným prvkem v závorkách bude operátor), vyhodí výjimku, kterou ošetří volající kód a informuje uživatele o chybě v zápisu výrazu. Kdybychom měli pomocí přepisovacích pravidel definovaných v programu zjednodušovat dříve zmíněný výraz `(x or (true and y)) or z`, bude to vypadat takto:

1. `true and y` bude zjednodušeno na `y`, a jelikož by výsledný výraz byl `(x or (y)) or z`, kde y je jediným prvkem v závorkách, budou odstraněny závorky a výsledným výrazem po této operaci bude `(x or y) or z`.

2. Pomocí definovaných pravidel již není možné výraz dále upravit, zjednodušování končí a výsledek po úpravě tedy zůstává `(x or y) or z`.

Je možné, že během zjednodušování výrazu byla úplně eliminována nějaká proměnná (bylo by tomu tak např. ve výrazu `x or not x`). Z toho důvodu, jakmile je výraz zjednodušen, zjistíme znovu všechny proměnné, které ve výrazu ještě zbyly. Nyní je nutné projít všechny kombinace ohodnocení těchto proměnných. Začneme tím, že první ze zbývajících proměnných nahradíme ve výrazu konstantou `true`, respektive `false`. Pak můžeme opět výraz zjednodušit. Po zjednodušení může nastat situace, kdy výraz nabývá hodnoty `true` nebo `false` - v takovém případě již víme, jestli je při určitém ohodnocení proměnné/proměnných výraz pravdivý (`true`) nebo nepravdivý (`false`). V opačném případě řešíme opět problém ohodnotit další první proměnnou hodnotami `true` a `false` apod. - postupujeme rekurzivně. Během tohoto postupu je nezbytné si nějakým způso-

bem ukládat, jak byly ohodnoceny předchozí proměnné. V programu je tento problém řešen tak, že funkce pro ohodnocení jedné proměnné vrací seznam: `[název proměnné, ohodnocení pokud je false, ohodnocení pokud je true]`, kde `ohodnocení pokud je false`, respektive `ohodnocení pokud je true` je True v případě, kdy pokud proměnné přiřadíme hodnotu true/false, formule bude pravdivá, nebo seznam v případě, kdy je nezbytné, aby následující proměnné nabývaly nějakých hodnot, případně False, pokud formule nebude pravdivá. Takový seznam může pro dříve zjednodušenou formuli `(x or y) or z` vypadat následovně:

```
['x', ['y', ['z', False, True], True], True]
```
(3.1)

Můžeme tak zjistit, že pokud `x` je true, celá formule bude pravdivá, jelikož na 2. indexu (číslováno od nuly) seznamu s proměnnou `x` je `True`. V případě, kdy `x` je false (na 1. indexu), buď musí být `y` true (2. index pro proměnnou `y` pokud `x` je false), nebo musí být true `z`. Takové chování přesně od naší formule očekáváme. Z těchto výsledků je již možné sestavit uživateli prezentovatelný závěr s potřebnými ohodnoceními proměnných tak, aby byl výraz pravdivý.

4 Programová dokumentace

Program má strukturu znalostního systému. Skládá se z těchto modulů: *main*, *baze_dat*, *baze_znalosti*, *inferencni_modul*, *komunikacni_modul* a *vysvetlovaci_modul*. Jako programovací jazyk byl zvolen Python, který umožňuje zápis vysokoúrovňového kódu včetně možnosti objektově orientovaného programování, je nezávislý na operačním systému, jednoduchý pro instalaci a kód psaný v tomto jazyce je relativně snadno čitelný. Data jsou napříč programem uchovávána převážně v seznamech nebo v objektech tříd. Tyto datové struktury/typy byly vždy zvoleny s cílem co nejvíce zjednodušit manipulaci s daty.

4.1 Modul main

Tento modul slouží ke spuštění programu. Pouze předává řízení inferenčnímu modulu a jakmile program skončí, čeká na ukončení uživatelem pomocí klávesy Enter.

4.2 Báze dat

Báze dat slouží k ukládání dat, v případě tohoto programu jsou již při spuštění v bázi dat dostupné předdefinované prvky logických výrazů (jako operátory, nebo konstanty). Za běhu programu jsou pak do báze dat přidávány uživatelem definované proměnné ve výrazu (po vyhodnocení jednoho výrazu jsou opět vymazány). Báze dat také obsahuje definici třídy `PrvekVyrazu`, která umožňuje jednodušší uložení dat.

4.3 Báze znalostí

Báze znalostí obsahuje přepisovací pravidla, pomocí kterých je možné zjednodušovat logický výraz. Celkem je definováno více než 50 přepisovacích pravidel, nicméně je možné v budoucnu tato pravidla rozšířit a vytvořit tak komplexnější program. Pro přepisovací pravidla definuje báze znalostí třídu `PrepisujeSeNa`, a jednotlivé pravidla jsou pak instancí této třídy. Příkladem definice přepisovacího pravidla může být například `PrepisujeSeNa([TRUE, AND, TRUE], [TRUE])`, které značí, že logický výraz `true and true` je možné přepsat na `true`. Báze znalostí pak obsahuje seznam všech přepisovacích pravidel *prepisovaci_pravidla*, ke kterým může přistupovat zejména inferenční modul.

4.4 Inferenční modul

V inferenčním modulu probíhá vyhodnocování splnitelnosti výrazu. Ve smyčce se dotazuje komunikačního modulu na zadaný příkaz, a ten poté vyhodnocuje. Vyhodnocování logických výrazů probíhá podle výše popsaného algoritmu (3. kapitola). Za zmínku v inferenčním modulu stojí některé funkce, například *ziskej_promenne_z_formule*, která z logického výrazu získá všechny jedinečné proměnné. Funkce *nahradit* se zase pokouší aplikovat jedno konkrétní přepisovací pravidlo na seznam a nahradit tak nějakou sekvenci prvků výrazu jinou. Funkce *ziskej_zanorene_vyrazy* pomocí rekurze dokáže z řetězce typu `"x or (true and y)"` získat seznam, jako například `[x, or, [true, and, y]]`. Funkce *ziskej_vysledek* maximálně zjednoduší logický výraz, a získá výsledek (konstantu *true* nebo *false*, nebo jiný výraz, ve kterém bude nutné nahradit některé proměnné konstantami). Funkce *vyhodnot_formuli* slouží k definitivnímu vyhodnocení splnitelnosti formule, a to metodami zjednodušování formule a postupným nahrazováním proměnných za konstanty.

4.5 Komunikační modul

Komunikační modul se stará o jednoduché výpisy na obrazovku. Kromě toho řeší IO operace, což v tomto programu znamená čtení logických výrazů ze souboru. Dále zprostředkovává komunikaci ostatních modulů. Mezi pár důležitých funkcí tohoto modulu by bylo možné zařadit *zeptej_se_na_vyraz* (zeptá se uživatele na vstup), *zobraz_napovedu* (zobrazí nápovědu), nebo *ziskej_radky_souboru* (načte všechny řádky ze souboru, ve kterém jsou uloženy logické výrazy).

4.6 Vysvětlovací modul

Vysvětlovací modul má na starost vysvětlení výsledků, které mu předá inferenční modul (přes komunikační) v podobě, která byla zmíněna v popisu algoritmu, tedy např.

```
['x', ['y', ['z', False, True], True], True]
```
(4.1)

Vysvětlovací modul pak musí takový výsledek srozumitelně vysvětlit - pokud je z inferenčního modulu vrácena pouze konstanta True/False, je jasné, že výraz je vždy pravdivý nebo vždy nepravdivý, v opačném případě vysvětlovací modul vypíše (mimo jiné) pravdivostní tabulku. Vzhledem k tomu, jakým způsobem poskytuje inferenční modul výsledky, je možné, že došlo k redukci nějaké proměnné během vyhodnocování určité větve výrazu, ale výraz jako celek zůstává na této proměnné závislý. Aby si s tímto problémem program poradil, je definována funkce *vypis_vsechny_varianty*, která vypíše danou větvi výsledku i s těmito proměnnými. O korektní vysvětlení se dále starají funkce *vysvetli_vysledek* a *vypis_promenne*.

5 Uživatelská dokumentace

Program spusťte pomocí příkazu `python main.py`, nebo obdobným způsobem (např. dvojitým kliknutím na soubor main.py). Budou Vám vypsány všechny příkazy, pomocí kterých můžete ovládat program a bude očekáván Váš vstup. Pomocí příkazu `napoveda` zobrazíte detailní nápovědu k jednotlivým příkazům, včetně ukázek použití. Pomocí příkazu `konec` můžete program ukončit. Pokud zadáte příkaz `vysledek <vyraz>`, například `vysledek x or y or z`, program určí splnitelnost zadaného výrazu. Můžete také zadat `nacist <soubor>`, díky čemuž bude každý řádek zadaného souboru interpretován jako jeden výraz a postupně se budou určovat všechny splnitelnosti takto definovaných výrazů v určeném souboru. Po každém vyhodnocení výrazu budete vyzváni k stisknutí

klávesy Enter, čímž se začne vyhodnocovat další výraz. Pro testování tohoto příkazu je u práce přiložen soubor `vstup.txt` s testovacími daty.

Program přímo nepodporuje prioritu logických operací, to je nicméně možné vyřešit přidáním nezbytných závorek.

Pokud se nepovede otevřít nebo načíst soubor, na výstupu se zobrazí chybová hláška a uživatel může zadávat nový vstup. Pro každý vyhodnocovaný výraz (ať už je načítán ze souboru nebo standardního vstupu) je pak vypsáno:

1. **True**, pokud je výraz vždy pravdivý,
2. **False**, pokud je výraz vždy nepravdivý,
3. **pravdivostní tabulka** s ohodnocením proměnných tak, že tyto kombinace vedou k pravdivosti výroku v ostatních případech.

6 Závěr

Práce efektivně řeší problém splnitelnosti logických formulí. Formule jsou během získávání výsledku mnohokrát zjednodušovány, což vede k celkem rychlé odezvě programu a dobrému uživatelskému zážitku. Implementace algoritmu pro řešení tohoto problému však nebyla triviální, a proto považuji dosažené výsledky za velmi dobré. Pokud by měl být program v budoucnu dále vyvíjen, bylo by vhodné implementovat prioritu logických operací, která je nyní řešena jen formou uzávorkování výrazu, nicméně v současné podobě řešení by tato úprava znamenala celkem zásadní změny struktury programu.

Program je uživatelsky přívětivý, nabízí jednoduché a srozumitelné ovládání a celkově poskytuje dobrou podporu pro řešení problému splnitelnosti logických formulí.