Slovenská technická univerzita

Fakulta informatiky a informačných technológii

Ilkovičova 3, 812 19 Bratislava

Umelá inteligencia

Zadanie č. 4

Peter Berta

Cvičiaci: Ing. Ivan Kapustík

Študijný odbor: Informatika

Ročník: 2. Bc

Akademický rok: 2015/2016

1. Riešený problém

Úlohou je vytvoriť jednoduchý dopredný produkčný systém, s prípadnými rozšíreniami, napríklad o kladenie otázok používateľovi alebo vyhodnocovanie matematických výrazov.

Produkčný systém patrí medzi znalostné systémy, teda medzi systémy, ktoré so svojimi údajmi narábajú ako so znalosťami. Znalosti vyjadrujú nielen informácie o nejakom objekte, ale aj súvislosti medzi objektami, vlastnosti zvolených problémov a spôsoby hľadania ich riešenia. Znalostný systém je teda v najjednoduchšom prípade dvojica - program, ktorý dokáže všeobecne manipulovať so znalosťami a báza znalostí, ktorá opisuje problém a vzťahy, ktoré tam platia. Znalosti majú definovanú nejakú štruktúru a spôsob narábania s touto štruktúrou - to sa nazýva formalizmus reprezentácie znalostí. Program vie pracovať s týmto formalizmom, ale nesmie byť závislý od toho, aké znalosti spracováva, inak by to už nebol systém, kde riešenie úlohy je dané použitými údajmi..

2. Opis riešenia

Pre riešenie tohto problému je kľúčové korektne načítať všetky fakty a pravidlá zo vstupných súborov. Program by totiž nemal poznať konkrétnu štruktúru podmienok v pravidlách (okrem špeciálnych podmienok).

Keď inicializujem fakty a pravidlá zo vstupných súborov, použijem funkciu, ktorá mi pomôže vytvoriť kombinácie podmienok v pravidlách a faktov. Pre každú prvú podmienku v pravidle zavolám funkciu pre ďalšiu podmienku ale fakty zas prehľadávam od začiatku. Argumentom tejto funkcie sú aj premenné, ktoré sú vytvorené kombináciou predchádzajúcich podmienok a faktov.

Výsledkom je spájaný zoznam inštrukcií, ktoré je možné vykonať. Tieto inštrukcie ešte nie sú vyfiltrované. Filtrácia prebieha počas výberu inštrukcie, ktorú vykonáme.

Na jeden krok vykonáme jednu inštrukciu. Keď už nebudeme mať k dispozícii žiadnu inštrukciu, ktorú by sme mohli vykonať, tak skončíme.

3. Reprezentácia údajov problému, použitý algoritmus

Vstupné súbory prečítame pomocou pomocných funkcií:

```
struct Lines *fakty = init_file("fakty.txt");
struct rule *pravidla = init_rules("pravidla.txt");
```

Fakty sú uložené v štruktúre, ktorá obsahuje text faktu a odkaz na ďalší fakt. Je to teda spájaný zoznam faktov. Použil som spájaný zoznam pre jednoduché odkazovanie sa na nasledujúce fakty počas tvorby kombinácií:

```
struct Lines {
	char *text;
	struct Lines *next;
};
```

Pravidlá sú uložené tiež v spájanom zozname s tým rozdielom, že pravidlá obsahujú tri riadky textu: meno pravidla, podmienky pravidla a dôsledky pravidla. Tiež je tu odkaz na nasledujúce pravidlo:

```
struct rule {
	char *name;
	char *condition;
	char *result;
	struct rule *next;
};
```

Premenné, ktoré sú tvorené kombináciou faktov a pravidiel sú uložené ako dvojica reťazcov. Jeden reťazec predstavuje meno premennej (‘?X’) a druhý reťazec predstavuje hodnotu premennej (‘Peter’). Tiež sú uložené v spájanom zozname, takže táto štruktúra rovnako obsahuje odkaz na nasledujúcu premennú:

```
struct variable {
	char *name, *value;
	struct variable *next;
};
```

Algoritmus

Ako som už skôr spomínal, pre každé pravidlo prejdeme cez všetky jeho podmienky, ktoré porovnáme so všetkými faktami. V prípade, že nájdeme vhodnú kombináciu faktov v rámci jedného pravidla, teda že sa nevyskytne konflikt medzi premennými, dôsledok daného pravidla pridáme do spájaného zoznamu s výsledkami:

```
pridaj_vysledok(dosad_var(dosledok, premenne));
```

Keď vyskúšam všetky dostupné kombinácie, budem prechádzať tento spájaný zoznam výsledkov, kým nenájdem takú inštrukciu, ktorá vykoná nejakú zmenu (pridanie faktu, ktorý vo faktoch ešte nie je). Ak takú inštrukciu nájdem, tak ju vykonám a vypíšem aktuálny stav faktov.

Teraz čakám na používateľa, kým stlačí ENTER. Pokračujem vo vykonávaní ďalšieho kroku, až kým nevyčerpám všetky možné relevantné inštrukcie.

Dosadenie

Táto funkcia slúži na doplnenie premenných do vzoru. Do ‘source’ sa dosadí hodnota ‘second’ namiesto mena ‘first’:

```
char *replace(char *source, char *first, char *second){…}
```

Túto funkciu využíva hlavne funkcia, ktorá dosadí do celého reťazca všetky premenné.

```
char *dosad_var(char *potom, struct variable *premenna){…}
```

Pridaj premennú

Táto funkcia pridá do zoznamu premenných premennú s menom ‘name’ a hodnotou ‘value’, pričom vracia číselnú hodnotu 1 alebo 0 podľa toho, či bolo pridanie do zoznamu úspešné, alebo nie. Ak sa daná premenná už v zozname nachádza a má rovnakú hodnotu, druhý krát už do zoznamu pridávaná nie je.

```
int pridaj_var(struct variable **var, char *name, char *value){…}
```

Uvoľňovanie

Na uvoľňovanie používam tieto dve funkcie:

```
struct Lines *uvolni_vysledky(){…}

struct variable *uvolni_premenne(struct variable *premenne){…}
```

Ich ‘return’ hodnota je NULL. Takto je to preto, aby som k danej premennej hneď aj nastavil hodnotu na NULL. Tiež to zvyšuje prehľadnosť kódu.

Vykonávanie inštrukcií

Nasledujúce funkcie používam na vykonanie jednotlivých inštrukcií:

```
struct Lines *vymaz_fakt(struct Lines *fakty, char *fact){…}

struct Lines *pridaj_fakt(struct Lines *fakty, char *fact){…}

struct Lines *sprava_fakt(struct Lines *fakty, char *fact){…}
```

4. Spôsob testovania a výsledky experimentov

Na testovanie správnosti riešenia a funkčnosti algoritmu som používal rôzne funkcie, pomocou ktorých som vypisoval spájané zoznamy.

Výstupy môjho programu som porovnával s očakávanými výstupmi pre rovnaké fakty a pravidlá. [1]

5. Zhodnotenie riešenia

Riešenie tohto problému v programovacom jazyku C možno nie je najoptimálnejšie riešenie, keďže existujú iné programovacie jazyky, ktoré podporujú prácu s reťazcami oveľa lepšie, ale na tento jazyk som už zvyknutý, a popravde som chcel aj vyskúšať, ako to pôjde.

Programovací jazyk C má výhodu, že je pre mňa jednoducho zrozumiteľný a jednoducho si viem napísať rôzne funkcie. Do budúcnosti by som však možno zvážil použitie iných programovacích jazykov, ako na príklad Java alebo C++.

6. Používateľská príručka

Vo funkcii *main()* sa dajú zmeniť mená vstupných súborov, ak existuje požiadavka použiť iné, ako tie základné.

Po spustení programu sa automaticky inicializuje prvý krok a hneď sa aj vykoná. Po ukončení každého kroku program čaká na vstup od používateľa. Stačí stlačiť tlačidlo ENTER.

Zdroje:

[1] Produkčný systém. Ing. Ivan Kapustík. http://www2.fiit.stuba.sk/~kapustik/dp-sys-ot.html