Mesterséges intelligencia 2.

gyakorlat

3. feladat

(kvantormentes formuláról eldönteni, hogy logikai törvény-e vagy sem)

Készítette:

Szathmáry László
III. PTM.

3. feladat:

Egy olyan program megírása, mely egy adott kvantormentes formuláról eldönti, hogy logikai törvény-e vagy sem.
Mivel a Gentzen-kalkulus helyes és teljes, ezért ha egy formula levezethető a Gentzen-kalkulusban, akkor a formula logikai törvény. Ha nem vezethető le: nem logikai törvény.

Megoldás:

Az első feladatot fogjuk kibővíteni, mégpedig azért, mert ott egy alkalmas reprezentációt használtunk a formula tárolására: formulafa. Vagyis az első feladat beolvasta a formulát, szintaktikailag leellenőrizte, majd fát épített belőle.
Most már elegendő csupán néhány olyan függvénnyel kiegészíteni ezt az első programot, amelyek közül az egyik megkapja a fa gyökerét, s a többi függvény segítségével eldönti, hogy a fában tárolt formula logikai törvény-e vagy sem.

Az első feladatban volt egy **Formula** osztály, ezt ki kell egészíteni pár függvénnyel:

```
class Formula {
  public:
    Formula() {};
    void vezerlo();
  private:
    Tipus tipus;
    Fuggveny fuggveny;
    Predikatum predikatum;
    Lista lista;
    char *formula;
    struct faelem *gyoker;

    void nyelv_megadasa();
    int  ellenoriz(int mit);
    void hiba() { cout<<"Mem.-foglalasi hiba.\n"; exit(-1); }
    void hiba(int kod, char betu='\0');
    int beker(int ebbe, int MAX=1);
    void filebol_epit();
    void fileba_ment();
    void formula_beolvasasa();
    char miez(char *mit);
    void szetszed();
    void eloellenorzes();
    void eloallit(char *&i, char atadando[]);
    struct faelem * epit(char *str, int lista_lesz=0);
    void bejar(struct faelem *gyoker);
    void utoellenorzes(struct faelem *gyoker);
    void operandus_lista(char *fv_pred_nev, char miez, char str[]);
    void itteni_lista(struct faelem *gyoker, char str[]);
    int log_torveny(struct faelem *gyoker);                    // ***** UJ *****
    int szekvent_ell(Verem bal, Verem jobb);                   // ***** UJ *****
};
```

Mint látható, csupán két új függvényünk van: a **log_torveny()** és a **szekvent_ell()**.

```
void Formula::vezerlo()
{
  clrscr();
  nyelv_megadasa();
  formula_beolvasasa();
  eloellenorzes();
  gyoker = epit(formula);
  printf("fa felepitese OK\n");
  getch();
  bejar(gyoker); putchar('\n');
  utoellenorzes(gyoker);
  printf("a fv.-ek es/vagy pred.-ok operandusainak tipusa(i) OK\n\n");
  if (log_torveny(gyoker)) printf("Logikai torveny!\n");          // ***** UJ *****
  else                     printf("Nem logikai torveny!\n");     // ***** UJ *****
  getch();
}
```

Ki kell még egészíteni a vezérlőt is: nem elég most már a fa felépítése: le is kell azt ellenőrizni, hogy logikai törvény-e vagy sem.
A **log_torveny()** megkapja a fa gyökerét, s majd alkalmas módon meghívja a **szekvent_ell()** függvényt.

```
int Formula::log_torveny(struct faelem *gyoker)
{
  Verem bal, jobb;

  jobb.berak(gyoker);
  return (szekvent_ell(bal, jobb));
}
```

Tehát a fv. megkapja a gyökeret, vagyis annak a címét. A Gentzen-kalkulusban egy formula alakja a következő (átalakítva szekventté): $\Gamma \rightarrow \Lambda$, ahol a "$\rightarrow$" jel bal és jobb oldalán formulákból álló multihalmaz áll.
Kezdetben, amikor kapunk egy formulát azt át kell alakítani szekventté: ez formálisan annyit jelent, hogy egy "$\rightarrow$"-at teszünk elé, azaz Γ üres, míg Λ-ban ott lesz maga a kiinduló formula.

A **Verem** osztályt azért tartottam meg, mert jó pár metódusára szükség lesz (pl. **berak()**). A Verem osztályt is ki kell majd egészíteni jó néhány fv.-nyel, s majd látható lesz, hogy ez már inkább lista, semmint verem, de mint már említettem a **Verem** osztály fv.-einek nagy része továbbra is hasznosítható, ezért nem hoztam létre egy újabb Lista osztályt.
Vagyis a **Verem bal, jobb;** végül is két lista lesz, amelyben mutatók lesznek, amelyek faelemekre mutatnak. A bal a Γ multihalmazt jelenti, mely kezdetben üres, a jobb pedig a Λ m.-halmaz, amelyben kezdetben ott lesz az egész formula. Ezt úgy tudjuk jelölni, hogy a "jobb" listába berakjuk a gyökeret.

A **szekvent_ell()** megkapja a "bal" és "jobb" listát, s egy logikai értékkel tér vissza aszerint, hogy logikai törvény-e vagy sem.

```
int Formula::szekvent_ell(Verem bal, Verem jobb)
{
  char log_jel;
  int  result;
  struct veremelem *balkeres, *jobbkeres;
  balkeres = bal.muvkeres(); jobbkeres = jobb.muvkeres();
  Verem bal_masol(bal), jobb_masol(jobb);
  struct veremelem *balkeres_masol  = bal_masol.muvkeres();
  struct veremelem *jobbkeres_masol = jobb_masol.muvkeres();

  if (balkeres)
  {
    log_jel = ((balkeres->fa)->nev)[0];    //a muveleti jel lekerdezese
    switch (log_jel)
    {
        case '&' :  bal.berak((balkeres->fa)->bal);
                    bal.berak((balkeres->fa)->jobb);
                    bal.torol(balkeres);
                    break;
        case '¬' :  jobb.berak((balkeres->fa)->jobb);
                    bal.torol(balkeres);
                    break;
        case '|' :  bal.berak((balkeres->fa)->bal);
                    bal.torol(balkeres);
                    result = szekvent_ell(bal, jobb);
                    if (!result) return 0;
                    //---------------------------------------
                    bal_masol.berak((balkeres_masol->fa)->jobb);
                    bal_masol.torol(balkeres_masol);
                    result = szekvent_ell(bal_masol, jobb_masol);
                    if (!result) return 0;
                    return 1;
        case '>' :  jobb.berak((balkeres->fa)->bal);
                    bal.torol(balkeres);
                    result = szekvent_ell(bal, jobb);
                    if (!result) return 0;
                    //---------------------------------------
                    bal_masol.berak((balkeres_masol->fa)->jobb);
                    bal_masol.torol(balkeres_masol);
                    result = szekvent_ell(bal_masol, jobb_masol);
                    if (!result) return 0;
                    return 1;
        case '=' :  bal.berak((balkeres->fa)->bal);
                    bal.berak((balkeres->fa)->jobb);
                    bal.torol(balkeres);
                    result = szekvent_ell(bal, jobb);
                    if (!result) return 0;
                    //---------------------------------------
                    jobb_masol.berak((balkeres_masol->fa)->bal);
                    jobb_masol.berak((balkeres_masol->fa)->jobb);
                    bal_masol.torol(balkeres_masol);
```

```
                    result = szekvent_ell(bal_masol, jobb_masol);
                    if (!result) return 0;
                    return 1;
    }//end of switch (log_jel)
  }//end of if (balkeres)
  if (jobbkeres)
  {
    log_jel = ((jobbkeres->fa)->nev)[0];    //a muveleti jel lekerdezese
    switch (log_jel)
    {
        case '|' :  jobb.berak((jobbkeres->fa)->bal);
                    jobb.berak((jobbkeres->fa)->jobb);
                    jobb.torol(jobbkeres);
                    break;
        case '>' :  bal.berak((jobbkeres->fa)->bal);
                    jobb.berak((jobbkeres->fa)->jobb);
                    jobb.torol(jobbkeres);
                    break;
        case '¬' :  bal.berak((jobbkeres->fa)->jobb);
                    jobb.torol(jobbkeres);
                    break;
        case '&' :  jobb.berak((jobbkeres->fa)->bal);
                    jobb.torol(jobbkeres);
                    result = szekvent_ell(bal, jobb);
                    if (!result) return 0;
                    //---------------------------------------
                    jobb_masol.berak((jobbkeres_masol->fa)->jobb);
                    jobb_masol.torol(jobbkeres_masol);
                    result = szekvent_ell(bal_masol, jobb_masol);
                    if (!result) return 0;
                    return 1;
        case '=' :  bal.berak((jobbkeres->fa)->bal);
                    jobb.berak((jobbkeres->fa)->jobb);
                    jobb.torol(jobbkeres);
                    result = szekvent_ell(bal, jobb);
                    if (!result) return 0;
                    //---------------------------------------
                    bal_masol.berak((jobbkeres_masol->fa)->jobb);
                    jobb_masol.berak((jobbkeres_masol->fa)->bal);
                    jobb_masol.torol(jobbkeres_masol);
                    result = szekvent_ell(bal_masol, jobb_masol);
                    if (!result) return 0;
                    return 1;
    }//end of switch (log_jel)
  }//end of if (jobbkeres)

  if (bal.van_egyezes(jobb)) return 1;              //2. axioma teljesul
  balkeres = bal.muvkeres(); jobbkeres = jobb.muvkeres();
  if (balkeres==NULL && jobbkeres==NULL) return 0;
  return (szekvent_ell(bal,jobb));
}
```

A függvény tehát kap 2 listát, a szekvent bal és jobb oldalát. Ezt kellene a Gentzen-kalkulus szabályai alapján átalakítani. A függvény rekurzív módon működik, s ha egy szabály alkalmazásakor kettéágazik a fa, akkor mindkét ágra be kell látni annak helyességét, vagyis mindkét ágra meg kell hívni ezt a rekurzív fv.-t. Ha mindkettő igaz, akkor az is igaz, amelyik szétágazott. Ha valamelyik hamis: a kezdeti formulánk nem log. törv.
Mivel van ágazási lehetőség, ezért a bal és jobb listáról készítünk egy másolatot (egy-egy olyan listát, amely ugyanazokat a mutatókat tartalmazza, amelyeket a "bal" és "jobb" listák). Erre azért van szükség, mert ágazásnál át kell alakítani az egyik ágat, s alkalmazni rá egy rekurzív hívást, hogy jó-e. Ennek az átalakításnak azonban nem szabad kihatnia a másik ágra, ezért ott a másolatot alakítjuk a megfelelő módon, s arra hívjuk meg a fv.-t rekurzíve. (Ezért a másolásért a **Verem** osztály második konstruktora a felelős, amely egy listát kap: erről kell másolatot csinálnia).

Minden műveleti jelre van 2 szabály a Gentzen-kalkulusban, ezért bal és jobb oldalt is keresünk egy műveleti jelet. A sorrend lényegtelen (multihalmazról lévén szó), ill. ha nem találunk műveletet, akkor már csak predikátumszimbólumok vannak azon az oldalon, vagy semmi (üres az az oldal).

Ha vmelyik oldalon találunk egy műveleti jelet, akkor megnézzük, hogy az mi: ÉS (&), NEGÁCIÓ (¬), VAGY (|), IMPLIKÁCIÓ (>), EKVIVALENCIA (=). Attól függően, hogy melyik oldalon találtuk, s milyen is az a jel átrendezzük a listákat a megfelelő módon, s ismét leellenőrizzük az új szekventünket (rekurzió). Ágazás esetén 2 hívás is lesz!

Ha bal és jobb oldalon is lerendeztük az elsőként megtalált műveleti jelet, akkor megnézzük, hogy bal és jobb oldalon van-e egyezés. Ha igen: teljesül a 2. axióma, vagyis logikai törvénnyel van dolgunk.
Ha nincs egyezés, s bal és jobb oldalon sincs műveleti jel, akkor már nem is tudunk rajta mit tovább alakítani ⇒ nem logikai törvény, 0-val tér vissza.
Ha ezen is túljut: volt meg műveleti jel, van mit még alakítani: újabb rekurzív hívás a bal és jobb oldalra.

```
class Verem {
 public:
   Verem() : fej(NULL), vege(NULL), meret(0) {};
   Verem(const Verem& v);
   ~Verem();
   void berak(struct faelem *cim);
   void berak(char mit);
   struct faelem * top();
   struct faelem * alja();
   struct faelem * kivesz();
   struct faelem * Verem::masodik();
   void osszefuz();
   void hiba(int kod);
   int meret;
   int van_benne(char *mi);                  // ***** UJ *****
   int van_egyezes(Verem v);                 // ***** UJ *****
   void torol(struct veremelem *mit);        // ***** UJ *****
   struct veremelem * muvkeres();            // ***** UJ *****
 private:
   struct veremelem *fej, *vege;
};
```

Következzék itt a **Verem** osztály kiegészítése:

```
Verem::Verem(const Verem& v) : fej(NULL), vege(NULL), meret(0)
{
  struct veremelem *akt = v.fej;
  while (akt)
  {
    berak(akt->fa);
    akt = akt->kov;
  }
}
```

Ez egy új konstruktor, amely kap egy listát, s arról egy másolatot készít.

```
struct veremelem * Verem::muvkeres()
{
  struct veremelem *akt = fej;
  Lista lista;

  char betu;
  while (akt)
  {
    betu = lista.miez(((akt->fa)->nev));
    if (betu=='l') return akt;
    akt = akt->kov;
  }
  return NULL;
}
```

Ez a fv. a listában megy végig, s keresi az első műveleti jelet ('l' → logikai műveleti jelet jelent). Vagy ennek a mutatójával tér vissza, vagy NULL-al, ha nincs műveleti jel.

```
void Verem::torol(struct veremelem *mit)
{
  if (meret==1)
  {
    free(fej);
    fej=vege=NULL;
    meret=0;
    return;
  }
  struct veremelem *elozo=NULL, *akt = fej;
  while (!(akt == mit))
  {
    elozo=akt;
    akt = akt->kov;
  }
  if (mit==vege)
  {
    vege=elozo;
    elozo->kov=NULL;
```

```
    free(akt);
    --meret;
    return;
  }
  if (!elozo)
  {
    fej=akt->kov;
    free(akt);
    --meret;
  }
  else
  {
    elozo->kov = akt->kov;
    free(akt);
  }
}
```

Ez csupán egy elemet kitöröl a veremből (itt van egy kis lista-beütése a dolognak). Kap egy mutatót, amely a törlendő elemre mutat, s azt kell eltávolítani.

```
int Verem::van_egyezes(Verem v)
{
  struct veremelem *akt = fej;

  char betu = ((akt->fa)->nev)[0];
  while (akt)
  {
    if (isupper(betu))                                    //vagyis predikatum
        if ( v.van_benne((akt->fa)->nev) ) return 1;
    akt=akt->kov;
  }
  return 0;
}

int Verem::van_benne(char *mi)
{
  struct veremelem *akt = fej;

  while (akt)
  {
    if (strcmp(mi,(akt->fa)->nev)==0) return 1;
    akt=akt->kov;
  }
  return 0;
}
```

Ez a két függvény azt nézi meg, hogy őbenne és az átadott listában van-e azonos elem. Az első fv. megy a listában, s ha talál egy predikátumot, akkor ezt átadja a másodiknak, amelyik megnézi, hogy a másik listában szerepel-e.

Futási eredmény:

Adjon meg egy elsorendu mat.log. nyelvet!
Szeretne file-bol betolteni egy nyelvet? i
A file neve: p

/*** ahol "p" tartalma:

```
Srt/Cnst
p(xl)

Fv

Pred
A()
B()
C()
D()

__END__
```

vagyis: megadunk 4 predikátumszimbólumot: A, B, C, D
***/

A formula:

A>(B>A)
	logikai törvény
(A>(B>C))>((A>B)>(A>C))
	logikai törvény
A>(B>(A&B))
	logikai törvény
(A&B)>A
	logikai törvény
(A&B)>B
	logikai törvény
(A>C)>((B>C)>((A|B)>C))
	logikai törvény
A>(A|B)
	logikai törvény
B>(A|B)
	logikai törvény
(A>B)>((A>(¬ (B)))>(¬(A)))
	logikai törvény
(¬(¬(A)))>A
	logikai törvény

(¬(A)) &A
	nem logikai törvény