Cajtinova transformacija

- seminarski rad -

Autori:	Samardžija Miloš i Milićević Đorđe
Mentor:	Marić Filip
Predmet:	Automatsko rezonovanje, 2017. godine
Ustanova:	Univerzitet u Beogradu, Matematički fakultet

Kratak uvod

Transformisanjem formule u konjuktivnu normalnu formu *(KNF)* možemo dobiti formulu čija je složenost eksponencijalna u odnosu na složenost početne formule. Umesto uobičajenog algoritma prevođenja formule u konjuktivnu normalnu formu, u praksi se najčešće koristi Cajtinovo (*Tseytin, Tseitin*) kodiranje, koje je linearno u smislu vremena i prostora, ali uvodi nove promenljive, zbog čega rezultujuća formula nije logički ekvivalentna početnoj formuli, već je *slabo ekvivalentna:* početna formula je zadovoljiva ako i samo ako je zadovoljiva rezultujuća formula. Kažemo još da je dobijena formula *ekvizadovoljiva*. Dakle, koristeći Cajtinovu transformaciju, dobijamo ekvizadovoljivu formulu u konjuktivnoj normalnoj formu, koja je samo za konstantni faktor veća od polazne formule. Za primene je to obično dovoljno dobro.

Pokažimo na jednom primeru transformisanje iskazne formule u Cajtinovu konjuktivnu normalnu formu.

Neka je polazna iskazna formula sledećeg oblika:

$$(p \vee (q \wedge r)) \wedge (\neg p \vee \neg r)$$

Za sve podformule ćemo uvesti nove atome (iskazna slova):

$$(p \vee s_1) \wedge (\neg p \vee \neg r) \wedge (s_1 \Leftrightarrow q \wedge r)$$
$$s_2 \wedge (\neg p \vee \neg r) \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1)$$
$$s_2 \wedge s_3 \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1) \wedge (s_3 \Leftrightarrow \neg p \vee \neg r)$$
$$s_4 \wedge (s_1 \Leftrightarrow q \wedge r) \wedge (s_2 \Leftrightarrow p \vee s_1) \wedge (s_3 \Leftrightarrow \neg p \vee \neg r) \wedge (s_4 \Leftrightarrow s_2 \wedge s_3)$$

Ekvivalencije koje smo dobili se na uobičajen način prevode u konjuktivnu normalnu formu.

Može se pokazati da se svaki konjukt transformiše u konjuktivnu normalnu formu sa najviše četiri klauze, od kojih svaka ima najviše tri literala. Problem kod Cajtinove transformacije je što uvodi veliki broj novih promenljivih, ali se broj promenljivih i broj klauza mogu redukovati određenim tehnikama.

Detalji implementacije i upotreba programa

Program za prevođenje određene iskazne formule u Cajtinovu formu je napisan u programskom jeziku *C++*. Za leksičku analizu je korišćen alat *Flex*, dok je za parsiranje korišćen alat *Bison*.

Prevođenje programa se vrši komandom *make*, a brisanje fajlova nastalih u procesu kompilacije komandom *make clean*.

Iskazna formula koju želimo da transformišemo se unosi preko standardnog ulaza, nakon pokretanja programa komandom *./tseitin*. Po završeku unošenja formule, pritisnuti *';'*, a zatim taster *'Enter',* i ako je formula korektno unešena, program će na standardni izlaz ispisati formulu koju ste uneli i njenu Cajtinovu formu.

U *main* funkciji se nad parsiranom formulom poziva glavni (omotač) metod *tseitinTransformation*, koji vrši transformaciju nad početnom formulom i vraća njenu Cajtinovu formu. Sledi prikaz implementacije tog metoda:

```
Formula BaseFormula::tseitinTransformation()
{
      AtomSet as;
      Formula simpl = simplify()->pushNegation();
      simpl->getAtoms(as);
      Formula tmp = nullptr;
      Formula res = _tseitin(simpl, as, tmp);

      if(tmp.get() == nullptr)
          return res;
      else
          return make_shared<And>(res, tmp);
}
```

Ulazna formula se simplifikuje pozivom rekurzivne metode *simplify*, a zatim se uradi spuštanje negacija rekurzivnom metodom *pushNegation*.

Zatim se nad formulom poziva rekurzivni metod *_tseitin* koji sprovodi Cajtinovu transformaciju i vraća novogenerisane atome. Na slici ispod se nalazi prikaz implementacije tog metoda.

Na samom početku metoda nad formulom se poziva metod *isNATF* koji proverava da li je ulazna formula negacija, atom, konstanta *tačno* ili konstanta *netačno*. Ukoliko je to zadovoljeno, kao rezultat se vraća prosleđena formula i izlazi se iz metoda *_tseitin*. Ukoliko to nije zadovoljneno, metod *_tseitin* se rekurzivno poziva za levu i desnu podformulu ulazne formule, a kao rezultat se dobijaju dve nove formule. Generiše se novi atom sa jedinstvenim identifikatorom koji obezbeđuje funkcija *getUniqueId*. U *switch* naredbi se, na osnovu tipa ulazne formule, pravi nova formula koja predstavlja konkatenaciju formula dobijenih rekurzivnom primenom *_tseitin* metode na podformule ulazne formule. U zavisnosti od vrednosti trećeg parametra metode, *tmp*, formira se izlazna formula (taj parametar se koristi za generisanje definicionih ekvivalencija).

```
Formula BaseFormula::_tseitin(const Formula &f, AtomSet &as, Formula &tmp) const
{
      if (isNATF(f))
          return f;

      // apply transformation on subformulas
      Formula ts1 = _tseitin(((BinaryConnective*) f.get())->getOp1(), as, tmp);
      Formula ts2 = _tseitin(((BinaryConnective*) f.get())->getOp2(), as, tmp);

      // make new atom
      string id = getUniqueId(as);
      Formula atom = make_shared<Atom>(id);
      as.insert(id);

      Formula conn;
      switch (f->getType())
      {
      case T_AND:
            conn = make_shared<And>(ts1, ts2);
            break;
      case T_OR:
            conn = make_shared<Or>(ts1, ts2);
            break;
      case T_IMP:
            conn = make_shared<Imp>(ts1, ts2);
            break;
      case T_IFF:
            conn = make_shared<Iff>(ts1, ts2);
            break;
      }

      if (tmp.get() == nullptr)
          tmp = make_shared<Iff>(atom, conn);
      else
          tmp = make_shared<And>(tmp, make_shared<Iff>(atom, conn));

      return atom;
}
```