T-79.144 **Syksy 2002**
Logiikka tietotekniikassa: perusteet
Laskuharjoitus 12
(opetusmoniste, paketti 3, sovellukset)
26.1 – 2.12.2002

1. Olkoon annettuna seuraavat lauseet[1]:

1. $\forall x(\text{Close}(\text{m}, x) \rightarrow \text{Reach}(\text{m}, x)$,
2. $\forall x(\text{Tall}(x) \wedge \text{On}(\text{m}, x) \wedge \text{Under}(x, \text{b}) \rightarrow \text{Close}(\text{m}, \text{b}))$,
3. $\text{Tall}(\text{c})$,
4. $\text{Climb}(\text{m}, \text{c})$,
5. $\forall z\text{Moves}(\text{m}, \text{c}, z)$,
6. $\forall x(\text{Climb}(\text{m}, x) \rightarrow \text{On}(\text{m}, x)$,
7. $\forall x \forall y \forall z(\text{Moves}(x, y, z) \rightarrow \text{Close}(y, z) \vee \text{Under}(y, z))$ ja
8. $\neg\text{Close}(\text{c}, \text{b})$.

Formalisoinnissa on käytetty seuraavia predikaatteja ja vakioita:

1. $\text{Tall}(x)$ = "x on korkea",
2. $\text{Climb}(x, y)$ = "x voi kiivetä y:n päälle",
3. $\text{Close}(x, y)$ = "x on lähellä y:tä",
4. $\text{On}(x, y)$ = "x on y:n päällä",
5. $\text{Reach}(x, y)$ = "x ylettää y:hyn",
6. $\text{Under}(x, y)$ = "x on y:n alla",
7. $\text{Moves}(x, y, z)$ = "x (voi) siirtää y:n z:n lähelle"
8. m = "apina",
9. c = "tuoli" ja
10. b = "banaani".

[1]Esimerkki on kirjasta R.D. Dowsing, V.J. Rayward-Smith, C.D. Walter: A First Course in Formal Logic and its Applications in Computer Science

a) Mieti, mitä lausejoukon lauseet tarkoittavat.

b) Osoita resoluutiolla, että $\exists x\text{Reach}(m, x)$ on lausejoukon looginen seuraus.

c) Tutki resoluutiotodistuksen rakennetta. Mitä hyötyä siitä voisi olla tehtävän apinalle?

d) Suorita vastaava päättely `otter`-ohjelman avulla.

e) Voidaanko lausejoukko kääntää `PROLOG`-ohjelmaksi?

Ratk.

Valituista vakioista voidaan päätellä, että universumissamme esiintyvät apina, tuoli ja banaani. Predikaateista edelleen, että esineitä voidaan siirtää, erityisesti, että apina voi siirtää tuolia. Esinepareille annetaan lisäksi suhteita, esim. esine a on b:n lähellä, alla, jne. Lisäksi lause 2 ilmoittaa, että apina voi päästä banaanien lähellä, jos se on jonkun korkean esineen päällä ja ko. esine on banaanien alla. Oletettavasti siis yritetään kuvata tilannetta, jossa banaanit ovat korkealla (esim. katossa, puussa) ja apinan pitää suorittaa tiettyjä toimenpiteitä saadakseen mahansa kylläiseksi.

Kirjoitetaan `otter`-ohjelmalle syötetiedosto `monkey.otr`:

```
% Monkey example

set(binary_res).            % Resoluutiostrategia
assign(max_mem, 1500).      % 1.5 MB
assign(max_seconds, 1800).  % 30 min
set(free_all_mem).

formula_list(sos).

(all x (close(m,x) -> reach(m,x))).
(all x ((tall(x) & on(m,x) & under(x,b)) -> close(m,b))).
tall(c).
climb(m,c).
(all z moves(m,c,z)).
(all x (climb(m,x) -> on(m,x))).
(all x (all y (all z (moves(x,y,z) ->
         (close(y,z) | under(y,z)))))).
-close(c,b).
-(exists x reach(m,x)).
```

```
end_of_list.
```

Kutsutaan ohjelmaa komennolla `otter < monkey.otr > monkey.out`
Tulostiedostosta löytyy nyt mm. kaavojen klausuuliesitys:

```
list(sos).
1 [] -close(m,x) | reach(m,x).
2 [] -tall(x) | -on(m,x) | -under(x,b) | close(m,b).
3 [] tall(c).
4 [] climb(m,c).
5 [] moves(m,c,z).
6 [] -climb(m,x) | on(m,x).
7 [] -moves(x,y,z) | close(y,z) | under(y,z).
8 [] -close(c,b).
9 [] -reach(m,x).
end_of_list.
```

sekä binääriresoluutiolla haettu todistus tyhjälle klausuulille:

```
1 [] -close(m,x) | reach(m,x).
2 [] -tall(x) | -on(m,x) | -under(x,b) | close(m,b).
3 [] tall(c).
4 [] climb(m,c).
5 [] moves(m,c,z).
6 [] -climb(m,x) | on(m,x).
7 [] -moves(x,y,z) | close(y,z) | under(y,z).
8 [] -close(c,b).
9 [] -reach(m,x).
10 [binary,1,9] -close(m,x).
11 [binary,6,4] on(m,c).
12 [binary,7,5] close(c,x) | under(c,x).
15 [binary,12,8] under(c,b).
17 [binary,2,11] -tall(c) | -under(c,b) | close(m,b).
24 [binary,17,15] -tall(c) | close(m,b).
26 [binary,24,3] close(m,b).
27 [binary,26,10] .
```

Näinollen klausuulijoukko on toteutumaton ja $\exists x \mathrm{Reach}(m, x)$ seuraa loogisesti premisseistä. Otterin tekemästä todistuksesta voi havaita, että tehdyt resoluutioaskeleet vastaavat täsmälleen niitä toimenpiteitä, joita apinan tulee tehdä ylettyäkseen banaaneihin. Klausuuliesityksestä selviää, että kaikki premissit eivät ole esitettävissä Horn-klausuuleina, joten suora käännös `PROLOG`-ohjelmaksi ei onnistu.

2. Esitetään luonnolliset luvut $0, 1, 2, \ldots$ muuttujattomilla termeillä 0, $s(0)$, $s(s(0))$, ..., jotka rakentuvat vakiosymbolista 0 ja funktiosymbolista s, joka tulkitaan funktioksi $s(x) = x + 1$ luonnollisille luvuille x.

a) Tarkoittakoon predikaatit $J2(x)$, $J3(x)$ ja $J6(x)$ sitä, että luonnollinen luku x on jaollinen kahdella, kolmella ja kuudella. Määrittele nämä predikaatit predikaattilogiikan lausein siten, että predikaatin $J6$ määritelmä perustuu predikaattien $J2$ ja $J3$ määritelmiin.

b) Laadi a-kohdan määritelmien perusteella Otterin syötetiedosto ja osoita sen avulla, että jos luonnollinen luku x on kahdella ja kolmella jaollinen, niin luonnollinen luku $x+6$ on kuudella jaollinen.

c) Kirjoita määritelmät PROLOGin sääntöinä ja hae kuudella jaollisia luonnollisia lukuja. Samoin, hae luonnollinen luku x siten, että $x + 5$ on kuudella jaollinen.

Ratk.
Todetaan ensin perustapaukset, s.o. että 0 on kahdella ja kolmella jaollinen.

$$J2(0)$$
$$J3(0)$$

Edelleen, kuinka näistä päätellään jaollisuus suuremmille luvuille:

$$\forall x(J2(x) \rightarrow J2(s(s(x))))$$
$$\forall x(J3(x) \rightarrow J2(s(s(s(x)))))$$

Ja lopuksi määritellään kuudella jaollisuus:

$$\forall x(J2(x) \wedge J3(x) \rightarrow J6(x))$$

B - kohdassa pitää ohjelma kääntää Otterille ja esittää tehtävässä mainittu kysely.

```
set(auto).

formula_list(usable).

% tietämys
J2(0).
J3(0).
all x (J2(x) -> J2(s(s(x)))).
all x (J3(x) -> J3(s(s(s(x))))).
all x (J2(x) & J3(x) -> J6(x)).

% kyselyn negaatio
- all x (J2(x) & J3(x) -> J6(s(s(s(s(s(s(x))))))).

end_of_list.
```

T-79.144 **Syksy 2002**
Logiikka tietotekniikassa: perusteet
Laskuharjoitus 13, ratkaisut
(opetusmoniste, paketti 3, sovellukset)
3. – 9.12.2002

1. Käytä invariantteja osoittaaksesi, että seuraava C-ohjelma palauttaa annetun taulukon suurimman alkion (ohjelman argumentit, `a` = taulukko ja `size` = taulukon koko, (size > 0)).

```
int max(int a[], int size) {
    int m = a[0];
    while(i < size) {
        if (a[i] > m) m = a[i];
        i = i + 1;
    }
    return m;
}
```

Ratk.
Todistetaan induktiolla silmukan suorituskertojen lukumäärän (k) suhteen, että:

$$m_k = max\{a[0], \ldots, a[i_k]\} \text{ ja } i_k \leq size$$

- Silmukka suoritettu 0 kertaa. Tällöin $m_0 = a[0] = max\{a[0]\}$ ja $i = 0$ ($\leq size$).
- Päteköön ominaisuus k:n suorituskerran jälkeen.
- $m_{k+1} = max\{max\{a[0], \ldots a[k]\}, a[k+1]\} = max\{a[0], \ldots, a[k+1]\}$ ja silmukan ehdon perusteella $i_k < size$ eli $i_{k+1} \leq size$ pätee. Suoritus päättyy, kun $i < size$ tulee epätodeksi, eli kun $i = size$. Tällöin muuttujan m arvona on $max\{a[0], \ldots, a[size-1]\}$ ja $i \leq size$.

2. Osoita lauselogiikan ja semanttisen taulun avulla oheisten C:n ehtolausekkeiden ekvivalenssi.

a) `!(a < b || a > b)`

b) `a == b`

3. Osoita induktiolla, että oheinen ohjelma palauttaa arvon yksi, jos ja vain jos merkkijono a edeltää aidosti merkkijonoa b leksikografisessa järjestyksessä.

```
int comp(char *a, char *b) {
    if(*a && *b) {
        if(*a < *b) return 1;
        else if(*a > *b) return 0;
        a++; b++;
        return comp(a, b);
    }
    if (*b) return 1;
    return 0;
}
```

Ratk.
Todistetaan vahvalla induktiolla `comp`-rutiinille annettavien merkkijonojen pituuksien summan suhteen.

Perustapaus: Pituuksien summa on nolla. Tällöin ainoa mahdollisuus on, että molemmat merkkijonot ovat tyhjiä (NULL). Tällöin rutiini palauttaa arvon 0, mikä on oikein.

Induktio-oletus: Toimikoon rutiini kaikille pituuksien summille k:hon asti.

Induktioaskel: Jos a:n 1. kirjain on ennen b:n 1. kirjainta $\Rightarrow$ a edeltää b:tä aidosti. Tällöin ohjelma palauttaa arvon 1, mikä on oikein. Jos taas a:n 1. kirjain on b:n 1. kirjaimen jälkeen, niin a on b:n jälkeen muös leksikografisessa järjestyksessä. Ohjelmakin palauttaa tällöin nollan.

Jos kirjaimet ovat samat, niin sama tarkastelu pitää suorittaa toisesta kirjaimesta alkavan alimerkkijonon suhteen. Ohjelma toteuttaa tämän rekursiivisena kutsuna, joka toimii oikein induktio-oletuksen perusteella.

Edelleen, jos a on tyhjä ja b ei-tyhjä, pitää palauttaa arvo 1. Ohjelman ehtolauseke `(*a && *b)` evaluoituu epätodeksi, mutta `(*b)` todeksi ja ohjelma palauttaa arvon 1, mikä on oikein.