9 PÄÄTTELY PREDIKAATTILOGIIKASSA

- Jos predikaattilogiikan lauseista eliminoidaan kvanttorit, niin jäljelle jääneitä lauseita voidaan käsitellä propositiologiikan päättelymenetelmillä.
- Menetelmää kutsutaan *propositionalisoinniksi* (propositionalization)
- Eksistenssilauseet korvataan lauseilla joissa sidotut muuttujat on korvattu Skolem-vakioilla eli uusilla vakiosymboleilla, joita ei ole tietämyskannassa ennestään.
- Universaalilauseet korvataan kaikilla mahdollisilla lauseilla jotka saadaan kun muuttujat korvataan tietämyskannassa esiintyvillä vakio- ja funktiotermeillä eli perustermeillä (ground term), joissa ei esiinny muuttujia.

- Universaalikvantifioidun lauseen $\forall v\ \alpha$ muuttuja v voidaan korvata millä tahansa *perustermillä* (ground term), eli vakio- tai funktiotermillä, joka ei sisällä muuttujia.
- *Sijoitus* eli *substituutio* θ on sidontalista, jossa kullekin muuttujalle annetaan sen korvaava perustermi.
- Merk. $\alpha(\theta)$ on lause, joka saadaan kun sijoitusta θ sovelletaan lauseeseen α.
- Universaalikvanttorin eliminoimiseksi voimme päätellä

$$\frac{\forall v\ \alpha}{\alpha(\{\, v/g \,\})},$$

missä v on mikä tahansa muuttuja ja g mielivaltainen perustermi.

- Eksistenssikvanttorin eliminoimisessa muuttuja v korvataan *Skolem-vakiolla* k, joka ei ennestään esiinny tietämyskannassa

$$\frac{\exists v\ \alpha}{\alpha(\{\, v/k \,\})},$$

- Esimerkiksi voimme lauseesta $\exists x$ Isä$(Jussi) = x$ päätellä *instantiaation* Isä$(Jussi) = F_1$, missä F_1 on uusi vakio.
- Eliminoimalla eksistenssikvanttorit korvaamalla muuttujat Skolem-vakioilla ja universaalikvanttorit kaikilla mahdollisilla instantiaatioillaan, muuttuu tietämyskanta oleellisesti propositionaaliseksi.
- Muuttujattomat atomilauseet kuten Opiskelija(Jussi) ja Äiti(Jussi, Meeri) on vain nähtävä propositiosymboleina.
- Täten propositiologiikan päättelyä voidaan soveltaa predikaattilogiikan lauseisiin.

- Jos tietämyskannassa kuitenkin käytetään funktiosymboleita, niin mahdollisten korvaavien perustermien lkm on ääretön: Isä(Isä(Isä(Jussi))) jne
- Herbrandin kuuluisan tuloksen mukaan 1. kertaluvun teorian loogisilla seurauksilla on todistus propositionalisoidussa teoriassa, joka käsittelee vain äärellistä osaa siitä.
- Täten sisäkkäisiä funktioita voidaan käsitellä etenemällä ikään kuin leveyssuuntaisesti kokeilemalla ensin vakiotermejä, sitten yhteen kertaan sovellettuja funktioita jne kunnes todistus löytyy.
- Päättely on siis täydellistä.
- Analogia Turingin koneiden pysähtymisongelmaan kuitenkin osoittaa, että *ongelma on ratkeamaton*.
- Tarkemmin: *ongelma on osittain ratkeava*: jos lause on looginen seuraus, todistus löytyy, mutta muuten algoritmi voi jatkaa todistuksen etsintää ikuisesti.

SAMAISTUS L. UNIFIKAATIO

- Em. menetelmä ei kuitenkaan ole kovin tehokas, koska siinä universaalilauseista tuotetaan erilaisilla sijoituksilla monta ilmentymää vaikka usein vain yhtä tarvitaan todistuksessa.
- Universaalilauseiden kaikkien muuttujasijoitusten aukikirjoittaminen vaikuttaa turhalta. Parempi olisi käyttää vain sellaisia sijoituksia, joista on hyötyä todistamisessa.
- Jos meillä on sijoitus θ s.e. $p_i'(\theta) = p_i(\theta)$, kaikilla i, missä p_i' ja p_i ovat atomilauseita kuten myös q, niin voimme käyttää *yleistettyä Modus Ponensia*

$$\frac{p_1', \ldots, p_n',\ (p_1 \wedge \cdots \wedge p_n \rightarrow q)}{q(\theta)}$$

- Esim. lauseista Opiskelija$(Jussi)$, $\forall x$ Ahkera(x) ja $\forall y$ Opiskelija$(y) \wedge$ Ahkera$(y) \rightarrow$ Valmistuu2007(y) voimme päätellä Valmistuu2007$(Jussi)$ sidonnan $\{\, y/Jussi, x/Jussi \,\}$ perusteella

- Jätetään universaalikvanttorit merkitsemättä ja saadaan

$$\frac{\text{Opiskelija}(Jussi), \text{Ahkera}(x),\ (\text{Opiskelija}(y) \wedge\ \text{Ahkera}(y) \rightarrow \text{Valmistuu2007}(y))}{\text{Valmistuu2007}(y)(\{y/Jussi, x/Jussi\})}$$

- Yleistetty Modus Ponens on eheä päättelysääntö
- Samaan tapaan kuin yleistetty MP voidaan "korottaa" propositiologiikasta predikaattilogiikkaan, voidaan muokata myös eteen- ja taaksepäinketjutus sekä resoluutiosääntö.
- Päättelyalgoritmeissa keskeinen käsite on lauseiden *samaistus* (unification)
- Samaistusalgoritmi UNIFY palauttaa syötteenä annetut lauseet samaistavan sijoituksen, jos sellainen on olemassa

$$\text{UNIFY}(p, q) = \theta,\ \text{s.e. } p(\theta) = q(\theta)$$

muuten samaistus epäonnistuu (fail)

$$\begin{aligned}
&\text{UNIFY}(\text{Tuntee}(Jussi, x), \text{Tuntee}(Jussi, Jaana)) \\
&\quad = \{\, x/Jaana \,\} \\
&\text{UNIFY}(\text{Tuntee}(Jussi, x), \text{Tuntee}(y, Pekka)) \\
&\quad = \{\, x/Pekka, y/Jussi \,\} \\
&\text{UNIFY}(\text{Tuntee}(Jussi, x), \text{Tuntee}(y, \text{Äiti}(y))) \\
&\quad = \{\, y/Jussi, x/\text{Äiti}(Jussi) \,\} \\
&\text{UNIFY}(\text{Tuntee}(Jussi, x), \text{Tuntee}(x, Elina)) \\
&\quad = fail
\end{aligned}$$

- Viimeinen samaistus epäonnistuu, koska muuttujaa x ei voida samanaikaisesti sitoa sekä $Jussi$in että $Elina$an
- Koska muuttujat ovat universaalikvantifioituja, niin Tuntee$(x, Elina)$ tarkoittaa, että kaikki tuntevat $Elina$n
- Näin ollen samaistuksen pitäisi onnistua

- Ongelman edellä aiheuttaa saman muuttujanimen käyttö
- Toisaalta eri lauseiden muuttujanimillä ei ole merkitystä (ennen sidontaa) joten voidaan uudelleennimetä muuttujat: jälkimmäinen lausee korvataan sellaisella jossa x on korvattu muuttujalla jota ei ole käytetty ennestään, esim $x_1$7.
- Jos samastus voidaan tehdä usealla eri sijoituksella, valitaan se joka asettaa vähiten rajoituksia (sitomisia vakioihin) muuttujille, siis *yleisin samaistus* (most general unifier).

$$\begin{aligned}
&\text{UNIFY}(\text{Tuntee}(Jussi, x), \text{Tuntee}(y, z)) \\
&\quad \{\, y/Jussi, z/x \,\} \\
&\quad \{\, y/Jussi, x/Jussi, z/Jussi \,\}
\end{aligned}$$

- Yleisimmän unifioijan etsivä algoritmi on esitetty kirjan kuvassa 9.1: Kumpaakin termiä edetään ja tutkitaan rekursiivisesti samastaen aina kummassakin esiintyviä termejä mikäli mahdollista. Jos ei ole mahdollista, niin samastus epäonnistuu.
- Mikäli verrataan yksittäistä muuttujaa monimutkaiseen termiin, täytyy tehdä esiintymistarkistus (occur check), jossa katsotaan ettei muuttuja esiinny monimutkaisen termin sisällä. Jos esiintyy niin sijoitusta ei löydy ja samastus epäonnistuu.
- Esiintymistarkistuksen takia algoritmin aikavaatimus on neliöllinen, mutta asia voidaan tehdä lineaarisestikin monimutkaisemmilla algoritmeilla.
- Usein esiintymistarkistus tosin jätetään tekemättä kokonaan, mikä johtaa eheyden menettämiseen mutta ei usein ole ongelma käytännössä.

ETEENPÄINKETJUTUS

- Eteenpäinketjutus toimii, vastaavasti kuin propositionaalisessa tapauksessa, Hornin lauseille eli literaalien disjunktioille, joissa korkeintaan yksi literaali on positiivinen. Jokainen lause on siis atomilause tai implikaatio, jonka runko on positiivisten atomilauseiden konjunktio ja kärki on yksittäinen positiivinen atomilause

$$\text{Opiskelija}(Jussi)$$
$$\text{Ahkera}(x)$$
$$\text{Opiskelija}(y) \wedge \text{Ahkera}(y) \rightarrow \text{Valmistuu2007}(y)$$

- (Tarkkaan ottaen Hornin lause: korkeintaan yksi positiivinen literaali, definite clause: täsmälleen yksi positiivinen literaali)
- Literaalimuodossa universaalikvanttorit jätetään usein kirjoittamatta ja muuttujat vain oletetaan universaalisti kvantifioiduiksi.

- Kuten propositiologiikassa lähtien liikkeelle faktoista, soveltaen yleistettyä Modus Ponensia voidaan tehdä eteenpäin ketjuttavaa päättelyä: käydään läpi sääntöjä ja jos löytyy sellainen jonka ehdot täyttyvät, lisätään sen takajäsenen tietämyskantaan. Jatketaan kunnes kyselyn tavoitteenna ollut lause saadaan todistettua tai uusia faktoja ei saada lisättyä.
- Nyt on huolehdittava siitä, ettei "uusi" fakta ole pelkkä tunnetun faktan uudelleennimentä, esim.

$$\text{Pitää}(x, Karkki)\ \text{Pitää}(y, Karkki)$$

- Kun päästään vaiheeseen, jossa kaikki pääteltävät lauseet ovat jo tietämyskannassa, kantaa kutsutaan päättelyprosessin *kiintopisteeksi* (fixed point).

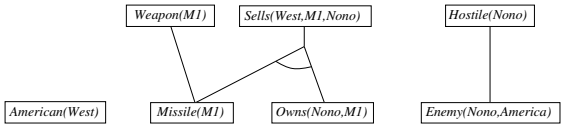
- Yleistetyn Modus Ponensin sovelluksena eteenpäin ketjutus on eheä päättelyalgoritmi. Se on myös täydellinen siinä mielessä, että tietämyskannan mitä tahansa loogista seurausta koskevaan kyselyyn saadaan vastaus.
- Datalog-tietämyskannassa ei esiinny laisinkaan funktiosymboleja ja eteenpäinketjutuksessa polynominen askelten lkm riittää kaikkien loogisten seurausten generoimiseen.
- Yleisessä tapauksessa joudumme vetoamaan Herbrandin tulokseen; päättely vain osittain ratkeavaa.
- Kirjan (amerikkalainen) esimerkki: USAn lain mukaan amerikkalaiset eivät saa myydä aseita vihamielisille valtioille. Valtio nimeltä Nono on vihollisvaltio, joka omistaa ohjuksia jotka on myynyt amerikkalainen eversti nimeltä West. Todistettava että eversti West on rikollinen.

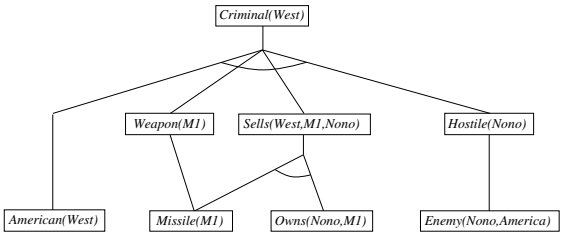
ESIMERKKI

- ... amerikkalaiset eivät saa myydä aseita vihamielisille valtioille:
  $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \rightarrow Criminal(x)$
- Nono omistaa ohjuksia ($\exists x\ Owns(Nono, x) \wedge Missile(x)$)
  $Owns(Nono, M_1)$ ja $Missile(M_1)$
- ... jotka on myynyt amerikkalainen eversti nimeltä West:
  $\forall x\ Missile(x) \wedge Owns(Nono, x) \rightarrow Sells(West, x, Nono)$
- Ohjukset ovat aseita:
  $Missile(x) \rightarrow Weapon(x)$
- Amerikan vihollliset ovat vihamielisiä:
  $Enemy(x, America) \rightarrow Hostile(x)$
- West on amerikkalainen:
  $American(West)$
- Nono on Amerikan vihollinen:
  $Enemy(Nono, America)$

function FOL-FC-ASK(*KB*, α) **returns** a substitution or *false*

repeat until *new* is empty
  new $\leftarrow \emptyset$
  for each sentence *r* **in** *KB* **do**
    $(p_1 \wedge \ldots \wedge p_n \implies q) \leftarrow$ STANDARDIZE-APART(*r*)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in *KB*
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence
          already in *KB* or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to *KB*
return *false*





- Eteenpäinketjutusalgoritmia käytetään nk. deduktiivisissa tietokannoissa (deductive databases) ja sääntöjärjestelmissä (production systems).
- Tämä algoritmi on kuitenkin tehoton kolmesta syystä:
  1. Algoritmin sisäsilmukassa generoidaan kaikki mahdolliset unifikaatiot, joilla säännön premissi saadaan samastettua sopivien faktojen kanssa.
  2. Jokainen sääntö tarkistetaan jokaisella iteraatiokerralla, vaikka vain yksikin uusi fakta olisi lisätty tietämyskantaan.
  3. Algoritmi generoi runsaasti faktoja, joilla ei ole merkitystä todistettavana olevan lauseen kannalta.
- Näitä ongelmia voidaan korjata eri menetelmillä ja parhaimmillaan eteenpäinketjutus toimii polynomisessa ajassa.

TAAKSEPÄINKETJUTUS

- Predikaattilogiikassa taaksepäinketjutuksessa maaliksi otetaan ensin todistettava lause ja katsotaan minkä säännön kärjen kanssa maali voidaan samaistaa.
- Kun tällainen sääntö löytyy, sen rungon kaikki konjunktit lisätään uusiksi maaleiksi ja jatketaan yhdistetyn sijoituksen (composition of substitutions) etsimistä jatkamalla sääntöjen läpikäyntiä uusilla maaleilla kunnes päästään faktoihin asti.
- Yhdistetty sijoitus määritellään perättäin sovelletuksi sijoitukseksi:
  $Subst(Compose(\theta_1, \theta_2), p) = Subst(\theta_2, Subst(\theta_1, p))$.
- Kun maali samaistuu faktaan, ei uusia (ali)maaleja tarvitse generoida
- Syvyyssuuntainen haku $\Rightarrow$ tilavaatimus on lineaarinen ja sen ongelmaksi tulee toistuvat tilat ja epätäydellisyys jos syvyys voi olla ääretön. Näitä ongelmia voidaan jossain määrin korjata.

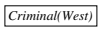
function FOL-BC-ASK(*KB*, *goals*, θ) **returns** a set of substitutions

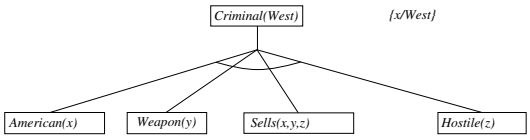
inputs: *KB*, a knowledge base
goals, a list of conjuncts forming a query (θ already applied)
θ, the current substitution, initially the empty substitution $\emptyset$

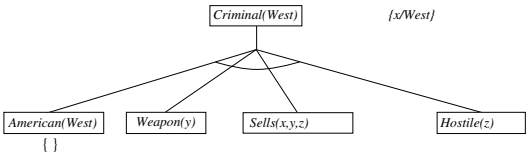
local variables: *answers*, a set of substitutions, initially empty

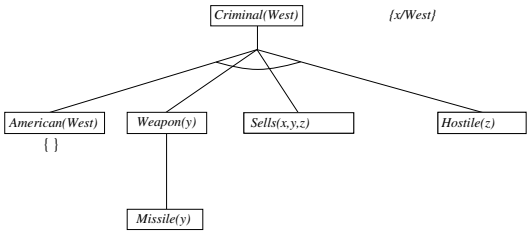
if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow$ SUBST(θ, FIRST(*goals*))
for each sentence *r* **in** *KB*
where STANDARDIZE-APART(*r*) = $(\ p_1 \ \wedge \ \ldots \ \wedge \ p_n \ \Rightarrow \ q)$
and $\theta' \leftarrow$ UNIFY(*q*, q') succeeds
new_goals $\leftarrow [\ p_1, \ldots, p_n | \text{REST}(goals)]$
answers $\leftarrow$ FOL-BC-ASK(*KB*, *new_goals*, COMPOSE(θ', θ)) $\cup$ *answers*
return *answers*

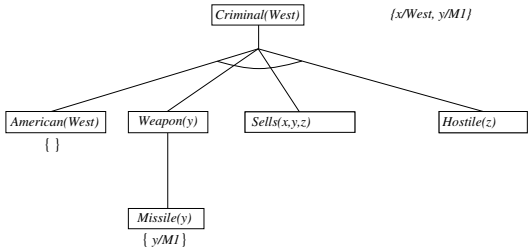
ESIMERKKI

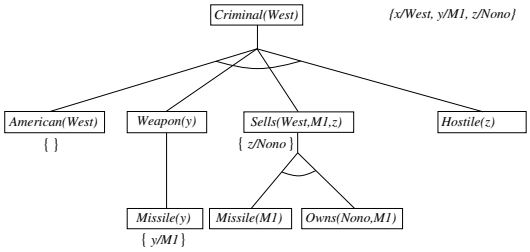


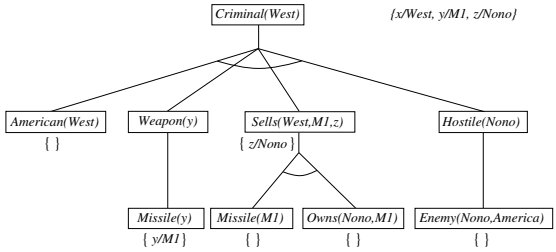












LOGIIKKAOHJELMOINTI

- Taaksepäinketjutus on logiikkaohjelmoinnin (logic programming) perusta. Logiikkaohjelmoinnin tavoitteena on deklaratiivinen tiedonesitys, jossa tietämys ja päättelyprosessit on erotettu toisistaan.
- Tunnetuin logiikkaohjelmointikieli on Prolog (Alain Colmerauer, 1972), jossa ohjelma muodostuu Hornin lauseina ilmaistusta tietämyskannasta. Siihen kohdistetaan kyselyitä, jotka sitten käsitellään taaksepäinketjutuksella.
- Prologin syntaksi on erilainen kuin logiikan: predikaattisymbolit alkavat pienellä kirjaimella ja muuttujat isolla, ensin kirjoitetaan klausuulin kärki, sitten "`:-`" (implikaatio oikealta vasemmalle) ja runko, missä literaalit on erotettu pilkulla (konjunktio) ja koko lause päättyy pisteeseen:

```
criminal(X) :- american(X), weapon(Y),
               sells(X,Y,Z), hostile(Z).
```


- Tämä siis vastaa predikaattilogiikan lausetta $\forall x, y, z$ American$(x) \wedge$ Weapon$(y) \wedge$ Sells$(x, y, z) \wedge$ Hostile$(z) \rightarrow$ Criminal(x)
- Prologissa on lisäksi määritelty operaatiot listojen käsittelyä ja aritmetiikkaa varten. Esimerkiksi listojen yhdistämiseen on määritelty predikaatti append(X,Y,Z), joka on tosi, kun Z on X:n ja Y:n yhdiste:

```
append([],Y,Y).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).
```

  Ensimmäisen mukaan Y saadaan yhdistämällä tyhjä lista Y:hyn ja toinen sanoo, että jos Z saadaan yhdistämällä X:ään Y, niin Z, jonka alusta on poistettu A saadaan yhdistämällä Y X:ään jonka alusta on poistettu A.
- Nyt voimme suorittaa kyselyn append([1],[2],Z), joka kysyy mitä saamme yhdistämällä listat [1] ja [2], ja saamme vastauksen Z=[1,2].

- Jos kyselyn maalina on lause append(A,B,[1,2]), siis mitkä kaksi listaa yhdistämällä saadaan lista [1,2], saadaan vastaukseksi mahdolliset muuttujasidonnat eli parit

```
A=[]    B=[1,2]
A=[1]   B=[2]
A=[1,2] B=[].
```

- Prolog-lauseiden evaluointi perustuu syvyyssuuntaiseen taaksepäinketjutukseen, missä lauseita kokeillaan siinä järjestyksessä, jossa ne on talletettu tietämyskantaan.
- Syvyyssuuntaisen päättelyn seurauksena voi olla ikuinen silmukka joten päättely ei ole täydellistä:

```
path(X,Z):- path(X,Y), link(Y,Z).
path(X,Z):- link(X,Z).
```

- Korjaus:

```
path(X,Z):- link(X,Z).
path(X,Z):- path(X,Y), link(Y,Z).
```


- Prologissa on joitain piirteitä joita ei ole tavallisessa loogisessa päättelyssä:
  - sisäänrakennetut aritmeettiset funktiot,
  - predikaateilla voi olla sivuvaikutuksia (esim. I/O),
  - negation as failure: negatiivinen predikaatti not P saa arvon true, jos positiivista predikaattia P ei onnistuta todistamaan. Esim. `alive(X) :- not dead(X)` joten kaikki joiden ei voi todistaa kuolleen ovat elossa.
  - Prologissa on yhtäsuuruusmerkki =, mutta se ei ole yhtä vahva kuin identiteettirelaatio logiikassa vaan toteutuu ainoastaan jos kaksi termiä voidaan samastaa. Ei ole mahdollista antaa faktoja tai sääntöjä joissa identtisyyttä olisi käytetty.
  - Kaikkia tarkistuksia (occur check) ei ole toteutettu Prologin unifikaatioalgoritmissa, joten päättely ei ole eheää. Tämä ei kuitenkaan yleensä johda ongelmiin.

RESOLUUTIO

- Kurt Gödelin *täydellisyyslause* (1930) 1. kertaluvun logiikalle: jokaisella loogisella seurauksella on äärellinen todistus
- $T \models \phi \Leftrightarrow T \vdash \phi$
- Vasta Robinsonin (1965) resoluutioalgoritmi antoi käytännöllisen menetelmän todistusten muodostamiseksi
- Gödelin tuloksista tunnetumpi on tietysti *epätäydellisyyslause*: matemaattisen induktion omaavan teorian kaikkia loogisia seurauksia ei voida todistaa aksioomista
- Erityisesti tämä koskee lukuteoriaa, jota siis ei voi aksiomatisoida

1. kertaluvun logiikan kaksi literaalia ovat komplementaarisia, jos yksi voidaan samaistaa toisen negaation kanssa

Resoluutiosääntö predikaattilogiikalle:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

missä $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

Esimerkiksi $\dfrac{\neg Rich(x) \vee Unhappy(x) \quad Rich(Ken)}{Unhappy(Ken)}$

missä $\theta = \{x/Ken\}$

Resoluutiossa premissien muuttujat täytyy tarvittaessa standardoida.

Johtopäätöksestä täytyy poistaa samaistuvat literaalit.

Resoluutiota varten lauseet on muutettava konjunktiiviseen normaalimuotoon. Esim.

$\forall x \ [\forall y \ \ \textit{Eläin}(y) \rightarrow \textit{Rakastaa}(x, y)] \rightarrow [\exists y \ \ \textit{Rakastaa}(y, x)]$

- **Implikaation poisto**

  $\forall x \ [\neg \forall y \ \neg \textit{Eläin}(y) \vee \textit{Rakastaa}(x, y)] \vee [\exists y \ \ \textit{Rakastaa}(y, x)]$

- **Negaation siirto sisään** $\neg \forall x, p \ \equiv \exists x \ \neg p$, $\neg \exists x, p \ \equiv \forall x \ \neg p$:

  $\forall x \ [\exists y \ \neg(\neg \textit{Eläin}(y) \vee \textit{Rakastaa}(x, y))] \vee$
  $[\exists y \ \ \textit{Rakastaa}(y, x)]$
  $\forall x \ [\exists y \ \neg\neg \textit{Eläin}(y) \wedge \neg \textit{Rakastaa}(x, y)] \vee [\exists y \ \ \textit{Rakastaa}(y, x)]$
  $\forall x \ [\exists y \ \ \textit{Eläin}(y) \wedge \neg \textit{Rakastaa}(x, y)] \vee [\exists y \ \ \textit{Rakastaa}(y, x)]$

- **Muuttujanimien standardointi**

  $\forall x \ [\exists y \ \ \textit{Eläin}(y) \wedge \neg \textit{Rakastaa}(x, y)] \vee [\exists z \ \ \textit{Rakastaa}(z, x)]$

- **Skolemisointi**

  $\forall x \ \ [\textit{Eläin}(F(x)) \wedge \neg \textit{Rakastaa}(x, F(x))] \vee \textit{Rakastaa}(G(x), x)$

- **Universaalikvanttorin poisto**

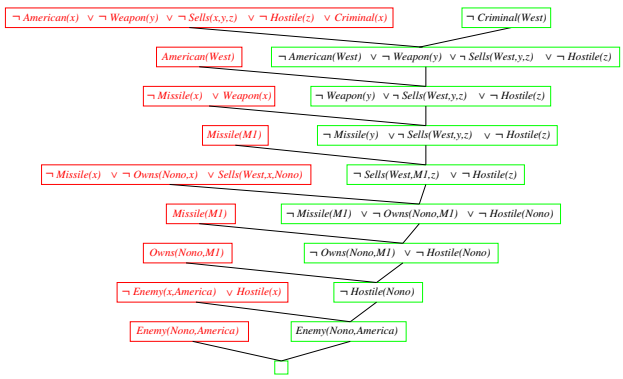
  $[\textit{Eläin}(F(x)) \wedge \neg \textit{Rakastaa}(x, F(x))] \vee \textit{Rakastaa}(G(x), x)$

- **Distributiivisuuden soveltaminen**

  $[\textit{Eläin}(F(x)) \vee \textit{Rakastaa}(G(x), x)] \wedge [\neg \textit{Rakastaa}(x, F(x)) \vee \textit{Rakastaa}(G(x), x)]$

- Lopputulos ei enää ole helposti ymmärrettävä, mutta se ei haittaa, koska muunnosproseduuri on automatisoitavissa
- Resoluutio on myös predikaattilogiikalle (refutaatio)täydellinen päättelyjärjestelmä siinä mielessä, että sen avulla voidaan tarkastaa tietämyskannan loogiset seuraukset
- KB $\models \alpha$ todistetaan osoittamalla, että KB $\wedge \neg\alpha$ johtaa ristiriitaan.

ESIMERKKI



- *Teoreemantodistimissa* logiikkaohjelmoinnin Hornin lauseisiin rajoitettu kieli on laajennettu täyteen 1. kertaluvun logiikkaan
- Myös esim. Prologin tapa antaa kontrollin sekoittaa lauseiden logiikkaa on yleensä poistettu
- Päättely ei riipu lauseiden tai niiden osien kirjoitusjärjestyksestä
- Sovelluskohde: ohjelmien ja laitteiston verifiointi
- Matematiikassa teoreemantodistimet ovat nousseet näkyvään asemaan
- Esimerkiksi 1996 Otter-ohjelman seuraaja ensimmäisenä todisti, että Robbinsin 1933 esittämät aksioomat määrittävät Boolen algebran