Teknillinen korkeakoulu

T-79.194
Tietojenkäsittelyteorian seminaari
10. huhtikuuta 2003

Stålmarckin todistusmenetelmä

Jori Dubrovin

48431A

Tiivistelmä. Stålmarckin menetelmällä voidaan ratkaista, onko mielivaltainen lauselogiikan lause pätevä vai ei. Menetelmä on kaupallisessa käytössä. Tässä esitellään Stålmarckin menetelmän periaatteet ja tärkeimmät ominaisuudet. Samalla pohditaan, miten menetelmä poikkeaa DPLL-pohjaisista toteutuvuustarkistusmenetelmistä.

Sisältö

1 Johdanto

Stålmarckin menetelmällä [1] voidaan ratkaista lauselogiikan pätevyysongelma: onko annettu lause F tosi kaikilla F:ssä esiintyvien muuttujien totuusarvoilla? Menetelmä on toteutettu kaupalliseen työkaluun, jota on käytetty järjestelmien verifiointiin teollisuudessa.

Kurssin aiheena on lauselogiikan toteutuvuustarkistimet konjunktiivisessa normaalimuodossa oleville lauseille. Toteutuvuus voidaan ratkaista myös yleiskäyttöisellä pätevyystarkistimella, sillä F on toteutuva jos ja vain jos $\neg F$ ei ole pätevä.

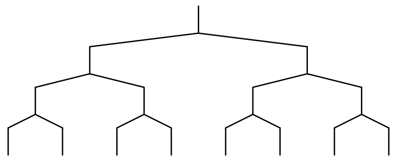
Menetelmässä yritetään löytää F:lle todistus eli osoittaa lause päteväksi tekemällä vastaoletus "F on epätosi jollakin muuttujien arvonjakelulla" ja käyttämällä päättelysääntöjä, kunnes päädytään ristiriitaan. Kun käytettävissä olevien päättelysääntöjen joukko on riittävä, tässä voidaan käyttää hyväksi *alilauseperiaatetta.* Periaatteen mukaan mille tahansa pätevälle lauseelle F on olemassa todistus, jossa päättelysääntöjä sovelletaan ainoastaan F:n alilauseisiin (myös $\top$ lasketaan alilauseeksi) tai niiden negaatioihin.

Päättelysääntöinä käytetään *yksinkertaisia sääntöjä* ja *Dilemma-sääntöä.* Yksinkertaisilla säännöillä saadaan tietoa muuttujista ilman haarautumista. Jos esimerkiksi tiedetään, että $x \wedge \neg y$ on tosi, voidaan päätellä, että x:n on oltava tosi ja y:n epätosi.

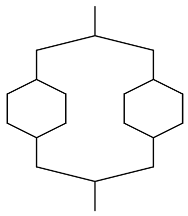
Yksinkertaiset säännöt eivät muodosta täydellistä todistusmenetelmää. Tarvitaan lisäksi jonkinlainen haarautumissääntö eli mahdollisuus spekuloida erilaisilla muuttujien totuusjakeluilla. Kokeillaan esimerkiksi mitä seuraa oletuksesta $x \equiv \top$, tai mitä jos olisikin $x \equiv \bot$. Monissa todistusmenetelmissä (mm. DPLL-menetelmät [3]) yritettäisiin johtaa ristiriita erikseen kummastakin oletuksesta. Tämä saattaa aiheuttaa sen, että todistuspuun koko räjähtää uusien haarautumisien myötä, tai todistus sisältää redundantteja osia (kuva 1).

Stålmarckin käyttämä Dilemma-sääntö on tavallaan hienostuneempi tapa toteuttaa haarautuminen. Haaroja ei yritetä ajaa loppuun saakka, vaan kummassakin haarassa tehdään vain ennalta määrätty määrä päättelyjä. Haarat kurotaan takaisin yhteen ja sen jälkeen voidaan mahdollisesti haarautua uudestaan jonkin toisen muuttujan suhteen. Dilemma-sääntöä käyttävä todistus saattaa olla kuvan 2 muotoinen.

Menetelmässä käytetyt tietorakenteet sekä yksinkertaiset säännöt esitellään luvussa 2. Luvussa 3 perehdytään tarkemmin Dilemma-sääntöön ja siihen perustuviin Dilemma-todistuksiin. Luvussa 4 esitetään algoritmi, joka etsii Dilemma-todistuksia, ja tutkitaan sen ominaisuuksia. Luvussa 5 katsotaan lyhyesti, mihin menetelmää on tähän mennessä sovellettu.



Kuva 1: Normaali todistuspuun muoto.



Kuva 2: Dilemma-sääntöön perustuva todistusgraafi.

2 Tietorakenteet ja yksinkertaiset säännöt

2.1 Kolmikot

Alkuperäinen todistettava lause on esitetty mielivaltaisena Boolen lausekkeena. Käsittelyn helpottamiseksi lause muutetaan kanoniseen muotoon, joka esitetään *kolmikoiden* avulla. Samalla saadaan tehokkaasti hyödynnettyä alilauseperiaatetta, joka esiteltiin luvussa 1.

Muunnostaulukkoa 1 käyttämällä annettu lause voidaan helposti muuttaa muotoon, joka ei sisällä muita konnektiiveja kuin $\wedge$ ja $\neg$. Tämän jälkeen kaikki F:n alilauseet ovat toisten alilauseiden konjunktioita tai niiden negaatioita. Varataan jokaista alilausetta kohti uusi muuttuja. Tällöin koko lauseen voi esittää konjunktiona kolmikoista, jotka ovat muotoa

$$x \equiv y \wedge z.$$

Tässä x, y ja z ovat joko todellisia muuttujia, alilauseita kuvaavia muuttujia, muuttujien negaatioita tai vakioita $\top$ tai $\bot$.

Jatkossa ei ole tarpeen erotella alkuperäisiä F:ssä esiintyviä muuttujia ja kolmi-

$$\begin{array}{rcl}
\phi \leftrightarrow \psi & \longmapsto & (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
\phi \rightarrow \psi & \longmapsto & \neg\phi \vee \psi \\
\phi \vee \psi & \longmapsto & \neg(\neg\phi \wedge \neg\psi) \\
\neg\neg\phi & \longmapsto & \phi
\end{array}$$

Taulukko 1: Muunnokset, joilla päästään eroon turhista konnektiiveista.

koita varten keksittyjä uusia muuttujia. Molempia kutsutaan lyhyesti muuttujiksi. *Literaali* tarkoittaa muuttujaa tai muuttujan negaatiota ja *vakio* symbolia $\top$ tai $\bot$.

Myös "alilauseelle" F varataan uusi muuttuja. Tämä kuvaa siis koko lauseen totuusarvoa.

Esimerkki. Lause

$$F = (a \overset{d}{\wedge} (b \overset{e}{\vee} c)) \overset{f}{\to} (\neg(\neg a \overset{g}{\vee} \neg b) \overset{h}{\vee} (a \overset{i}{\wedge} c))$$

voidaan muuntaa kolmikkomuotoon suoraankin muuntamatta muita konnektiiveja konjunktioiksi. Kutakin alilausetta edustava muuttuja on kirjoitettu vastaavan konnektiivin yläpuolelle. Koko lausetta edustaa muuttuja f. Lause kolmikkomuodossa:

$$\begin{array}{rcl} d \equiv a \wedge e, & \neg e \equiv \neg b \wedge \neg c, & \neg f \equiv d \wedge \neg h, \\ \neg g \equiv a \wedge b, & \neg h \equiv g \wedge \neg i, & i \equiv a \wedge c. \end{array}$$

Jos nyt haluaisimme osoittaa lauseen F päteväksi, riittäisi osoittaa, että ei ole olemassa sellaista muuttujien $a, b, \ldots, i$ arvonjakelua, joka toteuttaisi yllä olevat kuusi ekvivalenssia ja lisäksi vastaoletuksen $f \equiv \bot$.

2.2 Yksinkertaiset säännöt

Yksinkertaiset päättelysäännöt kuljettavat tietoa kolmikkojoukossa ilman haarautumista. Ne vastaavat DPLL-menetelmien [3] yksikköresoluutio- ym. sääntöjä. Kaikki yksinkertaiset säännöt on listattu alla. Symbolien x, y ja z paikalle voidaan kirjoittaa mikä tahansa literaali tai vakio.

$$\frac{x \equiv \neg x \wedge z}{\begin{array}{c} x/\bot \\ z/\bot \end{array}} \qquad \frac{x \equiv y \wedge \neg x}{\begin{array}{c} x/\bot \\ y/\bot \end{array}} \qquad \frac{x \equiv y \wedge y}{x/y} \qquad \frac{x \equiv y \wedge \neg y}{x/\bot} \qquad \frac{\top \equiv y \wedge z}{\begin{array}{c} y/\top \\ z/\top \end{array}}$$

$$\frac{x \equiv \top \wedge z}{x/z} \qquad \frac{x \equiv \bot \wedge z}{x/\bot} \qquad \frac{x \equiv y \wedge \top}{x/y} \qquad \frac{x \equiv y \wedge \bot}{x/\bot}$$

Nähdään, että säännön premissinä on aina yksittäinen kolmikko, jota kutsutaan säännön *laukaisevaksi* kolmikoksi. Muotoa x/y olevaa johtopäätöstä kutsutaan *assosiaatioksi.* Merkintä tarkoittaa, että x:llä ja y:llä on oltava sama totuusarvo. Johtopäätös voi olla ristiriitainen, esimerkiksi $(\neg a/a)$ tai $(b/\bot,\ \neg b/\bot)$. Tällöin voidaan todeta, että laukaiseva kolmikko on jo itsessään ristiriitainen.

Voi myös käydä niin että johtopäätös ei sisällä mitään informaatiota, kuten $\bot/\bot$. Tällöin kolmikko, johon sääntöä sovellettiin, on tosi kaikilla siinä esiintyvien muuttujien arvoilla, eikä kolmikon avulla saada mitään uutta tietoa.

Muussa tapauksessa assosiaatio x/y tarkoittaa, että jäljellä olevissa kolmikoissa voidaan samaistaa literaali (tai vakio) x literaalin (tai vakion) y kanssa. Käytännössä tämä tehdään joko korvaamalla kaikki x:n instanssit y:llä ja $\neg x$:n instanssit $\neg y$:llä tai päinvastoin. Joka tapauksessa säännön laukaissut kolmikko voidaan poistaa, koska säännön soveltamisen jälkeen kolmikko toteutuu triviaalisti.

Huomattakoon, että vaikka säännöt vaikuttavat itsestäänselviltä, saattavat x ja y edustaa mitä tahansa alilauseita tai niiden negaatioita. Näin ollen yksinkertaisten sääntöjen avulla voidaan löytää hyvinkin epätriviaaleja yhteyksiä alilauseiden välille.

Edellä olevat säännöt on lainattu Harrisonin implementaatiosta [2]. Harrison käyttää konjunktiivisten kolmikkojen $x \equiv y \wedge z$ lisäksi kolmikkoja, jotka ovat muotoa $x \equiv y \leftrightarrow z$. Tästä on se etu, että alkuperäisessä lauseessa olevia ekvivalensseja ei tarvitse muuntaa kolmeksi konjunktioksi, vaan ne voidaan suoraan kirjoittaa yhtenä kolmikkona. Tällaiset kolmikot olen yksinkertaisuuden vuoksi jättänyt pois. Stålmarckin omassa esityksessä [1] on tuotu esille muitakin mahdollisuuksia, tosin Stålmarck ei eksplisiittisesti kerro millaisia sääntöjä hän itse käyttää. Tämä liittynee siihen, että kyseessä on kaupallinen ohjelmisto.

2.3 Lauserelaatiot

Kaikki informaatio, jota todistuksen aikana kerätään, on muotoa x/y, eli kahdella literaalilla tai literaalilla ja vakiolla on sama totuusarvo. Jos käytössä olisi vain yksinkertaiset säännöt, voitaisiin assosiaatio x/y lukea korvaussääntönä "vaihda kaikkialla symboli x symboliksi y" ja unohtaa jatkossa kokonaan literaali x. Tätä tulkintaa käytettäessä tulee kuitenkin ongelmia siinä tilanteessa, kun Dilemma-säännön haarat pitäisi yhdistää.

Siksi määritellään literaalien ja vakioiden välinen ekvivalenssirelaatio R. Kaksi alkiota on samassa ekvivalenssiluokassa täsmälleen silloin, kun tiedetään, että niillä on sama totuusarvo. Tällaista relaatiota kutsutaan *lauserelaatioksi*. Jos R on lauserelaatio, niin ilmeisesti $(x, y) \in R$ jos ja vain jos $(\neg x, \neg y) \in R$. Lauserelaatio on *ristiriitainen*, jos jokin muuttuja ja sen negaatio (tai $\top$ ja $\bot$) ovat samassa ekvivalenssiluokassa.

Jos muuttujien totuusarvoista ei ole mitään tietoa, kukin literaali ja vakio muodostaa lauserelaatiossa oman ekvivalenssiluokkansa, jossa on vain yksi alkio. Jos saadaan selville, että x:llä ja y:llä on sama totuusarvo, yhdistetään ekvivalenssiluokat joihin x ja y kuuluvat. Samalla yhdistetään $\neg x$:n ja $\neg y$:n edustamat luokat. Yhdistäminen voidaan määritellä lauserelaatioiden unionin avulla.

Käytetään merkintää $\{x/y\}$ pienimmästä lauserelaatiosta R, jolle $(x, y) \in R$. Siis relaatiossa $\{x/y\}$ on ekvivalenssiluokat $\{x, y\}$ ja $\{\neg x, \neg y\}$. Loput luokat ovat jälleen yksialkioisia.

Lauserelaatioiden R_1 ja R_2 unioni $R_1 \cup R_2$ on pienin ekvivalenssirelaatio R, jolle pätee

$$(x, y) \in R_1 \text{ tai } (x, y) \in R_2 \quad \Longrightarrow \quad (x, y) \in R.$$

Näillä merkinnöillä $R \cup \{x/y\}$ tarkoittaa lauserelaatiota, joka saadaan R:stä sulauttamalla toisiinsa x:n ja y:n ekvivalenssiluokat ja toisaalta $\neg x$:n ja $\neg y$:n ekvivalenssiluokat. Voitaisiin sanoa, että $R \cup \{x/y\}$ saadaan yhdistämällä relaatio R ja assosiaatio x/y.

Mikä tahansa lauserelaatio saadaan unionina muotoa $\{x/y\}$ olevista relaatioista. Toisaalta lauserelaation R voidaan ajatella koostuvan kaikista niistä assosiaatioista x/y, joille pätee $(x, y) \in R$.

Määritellään vielä Dilemma-sääntöä varten lauserelaatioiden R_1 ja R_2 leikkaus. Se saadaan tavallisena joukkojen leikkauksena

$$(x, y) \in R_1 \cap R_2 \quad \Longleftrightarrow \quad (x, y) \in R_1 \text{ ja } (x, y) \in R_2.$$

2.4 Yksinkertaisten sääntöjen soveltaminen lauserelaatioihin

Yksinkertaisilla säännöillä tunnetuista kolmikoista ja assosiaatioista voidaan johtaa uusia assosiaatioita. Kun assosiaatiojoukot esitetään lauserelaatioiden avulla, voidaan ajatella, että yksinkertaisilla säännöillä johdetaan lauserelaatioista uusia lauserelaatioita.

Olkoot lähtökohtana ristiriidaton lauserelaatio R_1, kolmikko t ja yksinkertainen sääntö r. Ajatuksena on soveltaa sääntöä r kolmikkoon t, kun tiedetään, että relaation R_1 mukaiset ekvivalenssit ovat voimassa, eli jos $(x, y) \in R_1$, niin x:llä ja y:llä on sama totuusarvo. Johtopäätöksenä on uusi lauserelaatio R_2.

Valitaan kustakin R_1:n ekvivalenssiluokasta yksi edustaja. Tehdään tämä niin, että jos x:n ekvivalenssiluokan edustaja on y, niin $\neg x$:n ekvivalenssiluokan edustajaksi otetaan $\neg y$. Lisäksi vakiot $\top$ ja $\perp$ valitaan aina omien luokkiensa edustajiksi.

Muodostetaan uusi kolmikko t' niin, että otetaan kolmikko t ja korvataan kukin literaali sen ekvivalenssiluokan edustajalla, johon literaali kuuluu. Lopputulos on, että t' muuten sama kuin t, mutta joitakin literaaleja on mahdollisesti muutettu toisiksi ekvivalenteksi literaaleiksi tai vakioiksi niin, että t' sisältää enintään yhden edustajan kustakin R_1:n ekvivalenssiluokasta. Tämän sievennyksen jälkeen voidaan tutkia, soveltuuko sääntö r kolmikkoon t'.

Jos sääntöä ei voida soveltaa kolmikkoon, asetetaan $R_2 = R_1$. Muussa tapauksessa merkitään säännön r johtopäätöksenä olevia assosiaatioita a:lla ja b:llä (tai pelkästään a:lla jos assosiaatioita on vain yksi). Asetetaan $R_2 = R_1 \cup \{a\}$ tai $R_2 = R_1 \cup \{a\} \cup \{b\}$ sen mukaan, montako assosiaatiota johtopäätöksessä on.

Sanotaan, että näin saatu relaatio R_2 johdetaan relaatiosta R_1 ja kolmikosta t säännöllä r. Nähdään, että jos t' on sisäisesti ristiriitainen kolmikko, relaatiosta R_2 tulee ristiriitainen.

$$\frac{R}{\begin{array}{cc} R \cup \{x/y\} & R \cup \{x/\neg y\} \\ \text{(johto)} & \text{(johto)} \\ R_1 & R_2 \end{array}}{R_1 \sqcap R_2}$$

Taulukko 2: Dilemma-sääntö.

3 Dilemma-todistusjärjestelmä

3.1 Dilemma-sääntö

Dilemma-sääntö perustuu logiikan kaksiarvoisuuteen. Jos x ja y ovat kaksi literaalia tai vakiota, niin x:llä ja y:llä on joko sama totuusarvo tai vastakkainen totuusarvo. Näihin oletuksiin perustuvia päätelmiä ajetaan rinnakkain ja lopuksi tulokset yhdistetään. Tutkitaan sääntöä taulukon 2 avulla.

Säännön premissinä on lauserelaatio R. Relaatio R_1 saadaan liittämällä R:ään assosiaatio x/y ja käyttämällä joitakin päättelysääntöjä. Vastaavasti oletuksesta $x/\neg y$ saadaan sääntöjä käyttämällä johdettua lauserelaatio R_2. Nämä säännöt saattavat käyttää premisseinään paitsi lauserelaatioita, myös kolmikkoja, joita ei ole merkitty taulukossa näkyviin.

Johtopäätöksenä on lauserelaatio $R_1 \sqcap R_2$, joka määritellään seuraavasti.

$$R_1 \sqcap R_2 = \begin{cases} R_1, & \text{jos vain } R_2 \text{ on ristiriitainen,} \\ R_2, & \text{jos vain } R_1 \text{ on ristiriitainen,} \\ \{\top/\bot\}, & \text{jos } R_1 \text{ ja } R_2 \text{ ovat ristiriitaisia,} \\ R_1 \cap R_2, & \text{jos kumpikaan ei ole ristiriitainen.} \end{cases}$$

Relaation $\{\top/\bot\}$ paikalla voisi olla mikä tahansa muukin ristiriitainen relaatio, sillä ristiriitaisia lauserelaatioita ei ole koskaan tarpeen erotella toisistaan.

Säännön idea on se, että johtopäätös sisältää kaiken informaation, joka pitää paikkansa riippumatta siitä, oletetaanko x/y vai $x/\neg y$. Jos yksi oletus johtaa ristiriitaan, toisen oletuksen on pidettävä paikkansa. Jos molemmissa haaroissa päädytään ristiriitaan, päätellään että alkuperäinen relaatio R on ristiriitainen (tai ristiriidassa kolmikkojen kanssa, joita käytetään haarojen johdoissa).

Esimerkki. Olkoon vasemmassa haarassa oletuksena $a/\top$ ja oikeassa haarassa $a/\bot$. Vasemmassa haarassa johtopäätökset ovat $b/\neg c$ ja $c/\top$, ja oikeassa haarassa $b/\top$. Oletetaan vielä, että premissinä oleva relaatio R ei sisällä mitään tietoa muuttujista. Merkitään lauserelaatioita ekvivalenssiluokkien joukkoina. Tällöin lauserelaatiot olisivat taulukon 2 symbolein

$$R = \{\{\top\}, \{a\}, \{b\}, \{c\}, \{\bot\}, \{\neg a\}, \{\neg b\}, \{\neg c\}\},$$

$$R \cup \{a/\top\} = \{\{\top, a\}, \{b\}, \{c\}, \{\bot, \neg a\}, \{\neg b\}, \{\neg c\}\},$$

$$R_1 = R \cup \{a/\top\} \cup \{b/\neg c\} \cup \{c/\top\} = \{\{\top, a, \neg b, c\}, \{\bot, \neg a, b, \neg c\}\};$$

$$R \cup \{a/\bot\} = \{\{\top, \neg a\}, \{b\}, \{c\}, \{\bot, a\}, \{\neg b\}, \{\neg c\}\},$$

$$R_2 = R \cup \{a/\bot\} \cup \{b/\top\} = \{\{\top, \neg a, b\}, \{c\}, \{\bot, a.\neg b\}, \{\neg c\}\};$$

$$R_1 \sqcap R_2 = R_1 \cap R_2 = \{\{\top\}, \{a, \neg b\}, \{c\}, \{\bot\}, \{\neg a, b\}, \{\neg c\}\}.$$

Kuten relaatiosta $R_1 \sqcap R_2$ nähdään, Dilemma-säännöllä voidaan tässä tapauksessa päätellä, että a:lla ja b:llä on vastakkaiset totuusarvot.

3.2 Dilemma-todistukset

Dilemma-johto on päättelyketju, jossa on käytetty yksinkertaisia sääntöjä ja Dilemma-sääntöä. Se voidaan määritellä seuraavasti. Oletetaan, että kolmikkojoukko T on kiinnitetty.

1. *Triviaali johto.* Jos R on lauserelaatio, niin

   $$R$$

   on Dilemma-johto R:stä R:ään.

2. *Yksinkertaiset säännöt.* Olkoon Π Dilemma-johto R_1:stä R_2:een. Jos on olemassa kolmikko $t \in T$ ja yksinkertainen sääntö r siten, että säännöllä r voidaan johtaa relaatiosta R_2 ja kolmikosta t relaatio R_3, niin

   $$\begin{array}{c} \Pi \\ R_3 \end{array}$$

   on Dilemma-johto R_1:stä R_3:een.

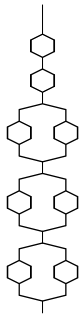
3. *Dilemma-sääntö.* Olkoon Π_0 Dilemma-johto R_0:sta R:ään ja olkoot x ja y literaaleja tai vakioita. Jos Π_1 on Dilemma-johto $R \cup \{x/y\}$:stä R_1:een ja Π_2 on Dilemma-johto $R \cup \{x/\neg y\}$:stä R_2:een, ja $R_3 = R_1 \sqcap R_2$, niin

   $$\begin{array}{c} \Pi_0 \\ \hline \Pi_1 \qquad\qquad \Pi_2 \\ \hline R_3 \end{array}$$

   on Dilemma-johto R_0:sta R_3:een.

4. Muita Dilemma-johtoja ei ole.

Dilemma-johdon *syvyys* on siinä käytettyjen sisäkkäisten Dilemma-sääntöjen maksimilukumäärä. Johdossa, jonka syvyys on 0, on käytetty ainoastaan yksinkertaisia sääntöjä. Jos johdon syvyys on 1, siinä on käytetty Dilemma-sääntöä,



Kuva 3: Dilemma-todistus, jonka syvyys on 2.

mutta kaikki Dilemma-sääntöjen haarat sisältävät vain yksinkertaisia sääntöjä, jne.

Määritellään, mikä on pätevän lauseen Dilemma-todistus. Olkoon F pätevä lause ja T sitä vastaava kolmikkojoukko. Merkitään f:llä muuttujaa, joka edustaa koko lausetta F. Jos Π on Dilemma-johto R_1:stä R_2:een, missä $R_1 = \{f/\bot\}$ ja R_2 on mikä tahansa ristiriitainen lauserelaatio, niin Π on F:n *Dilemma-todistus.*

Jos lause F on pätevä, sen *vaikeusaste* on pienin luku k siten, että F:lle on olemassa Dilemma-todistus, jonka syvyys on k. Kaikilla pätevillä lauseilla on äärellinen vaikeusaste: jos F:ssä esiintyvien muuttujien lukumäärä on m, voidaan periaatteessa helposti rakentaa m:n syvyinen todistus, jossa kokeillaan kaikki mahdolliset muuttujien arvonjakelut. Tämä vastaa F:n täydellisen totuustaulun rakentamista. Toisaalta kaikille m on olemassa päteviä lauseita, joiden vaikeusaste on vähintään m [1].

Vaikeusaste on käytännössä keskeinen käsite, kun etsitään todistusta F:lle. Jotta vältyttäisiin turhalta työltä, käytännössä k:n syvyistä todistusta aletaan etsiä vasta, kun kaikki mahdollinen on tehty $k-1$:n syvyisen todistuksen löytämiseksi. Tämän perusteella tyypillisen todistuksen graafiesitys voisi olla kuvan 3 näköinen.

4 Saturaatioalgoritmi

Stålmarckin menetelmän toteuttaminen perustuu saturaatioalgoritmiin, joka tekee täydellisen haun löytääkseen Dilemma-todistuksen, jonka syvyys on enintään k. Luku k on algoritmille annettava argumentti. Muut argumentit ovat alkuperäinen kolmikkojoukko sekä lauserelaatio, joka sisältää lähtöoletukset sekä mahdollisesti aikaisemmin hankitun tietämyksen.

Määritellään ensin, mitä saturaatiolla tarkoitetaan. Oletetaan edelleen, että kolmikkojoukko T on kiinnitetty. Ristiriidaton lauserelaatio R on *k-saturoitu*, jos ei ole olemassa Dilemma-johtoa R:stä R':uun siten, että johdon syvyys on enintään k ja $R \neq R'$. Toisin sanoen R ei ole k-saturoitu, jos R:stä voidaan johtaa uusia assosiaatioita Dilemma-johdolla, jonka syvyys on k. Tämän perusteella on helppo arvata, miten saturaatioalgoritmi toimii.

0-saturaatioalgoritmi soveltaa yksinkertaisia sääntöjä, kunnes niitä ei enää voi soveltaa, ja palauttaa näin saadun lauserelaation.

1-saturaatioalgoritmi valitsee muuttujan x ja soveltaa Dilemma-sääntöä siten, että haaroissa oletetaan $x/\top$ ja $x/\bot$, ja kummassakin haarassa käytetään vain yksinkertaisia sääntöjä. Tätä jatketaan, kunnes uusia assosiaatioita ei saada johdettua tällä tavoin. Algoritmi kokeilee siis kaikkia muuttujia uudestaan ja uudestaan, kunnes huomaa että kokonainen kierros on menty läpi ilman että on saatu uutta informaatiota.

2-saturaatio toimii muuten samoin, mutta haarautumisen jälkeen kummassakin haarassa suoritetaan 1-saturaatio, jonka jälkeen haarat yhdistetään. Vastaavasti $k+1$ -saturaatio voidaan määritellä k-saturaation avulla.

Saturaatioalgoritmi valitsee haarautumisehdossa (x/y vs. $x/\neg y$) aina $y = \top$, vaikka Dilemma-säännössä voisi periaatteessa ottaa y:n paikalle minkä tahansa literaalin. Tämä tapa on käytännössä koettu tehokkaaksi [1].

4.1 0-saturaatio

Nyt voimme koota yhteen aiemmin esitetyt ideat. 0-saturaatioalgoritmin pseudokoodi on esitetty alla.

```
   saturate(triplets T, relation R, depth 0) :=

1  Q := apply substitutions R on T
2  while (Q not empty)
3    q := some triplet from the set Q
4    Q := Q - {q}
5    if (a simple rule applicable to q)
6      R' := conclusions of the simple rule
7      if (R' == "contradiction")
8        return("contradiction")
9      end if
10     Q := Q union {triplets in T affected by R'}
11     Q := apply substitutions R' on Q
12     R := R union R'
13   end if
14 end while
15 if (R is a full truth assignment)
16   exit("countermodel found")
17 end if
18 return(R)
```

Funktio etsii melko suoraviivaisesti kaikki kolmikot, joihin voidaan soveltaa jotakin yksinkertaista sääntöä, ja kasvattaa lauserelaatiota R vastaavasti. Rivillä

1 kopioidaan kolmikot joukkoon Q ja samalla korvataan literaaleja ekvivalenssiluokkien edustajilla niin, että Q:ssa esiintyy enintään yhtä edustajaa kustakin R:n ekvivalenssiluokasta. Samoin rivillä 11 varmistetaan, että Q:ssa ei ole useita edustajia samasta ekvivalenssiluokasta. Rivillä 6 asetetaan R':uun relaatio, joka saadaan yksinkertaisen säännön johtopäätöksenä olevista assosiaatioista. Ehto `R' == "contradiction"` rivillä 7 on tosi, jos R' on mikä tahansa ristiriitainen lauserelaatio.

Aina kun yksinkertaista sääntöä on sovellettu, rivillä 10 laitetaan Q:hun takaisin ne kolmikot, joihin säännön soveltaminen vaikuttaa. Jos relaatio R antaa totuusarvon $\top$ tai $\bot$ jokaiselle muuttujalle eikä ole ristiriitainen, rivillä 16 keskeytetään todistuksen etsiminen, koska vastamalli on löytynyt.

4.2 $k+1$ -saturaatio

$k+1$ -saturaatioalgoritmi soveltaa Dilemma-sääntöä vuorotellen kaikkiin muuttujiin siten, että kummassakin haarassa suoritetaan k-saturaatio.

```
   saturate(triplets T, relation R, depth k+1) :=

 1 repeat
 2   V := set of variables in T
 3   V := apply substitutions R on V
 4   R' := R
 5   for (each v in V)
 6     R1 := saturate(T, R union {v/TRUE}, k)
 7     R2 := saturate(T, R union {v/FALSE}, k)
 8     if (R1 == "contradiction" && R2 == "contradiction")
 9       return("contradiction")
10     else if (R1 == "contradiction")
11       R := R2
12     else if (R2 == "contradiction")
13       R := R1
14     else
15       R := R1 intersect R2
16     end if
17   end for
18 until (R' == R)
19 return(R)
```

Riveillä 2 ja 3 asetetaan joukkoon V kaikki kolmikoissa esiintyvät muuttujat niin, että kustakin ekvivalenssiluokasta otetaan mukaan yksi edustaja. Rivillä 6 lauseke `R union {v/TRUE}` tarkoittaa, kuten aiemminkin määriteltiin, että lauserelaatiossa R yhdistetään ekvivalenssiluokat, joihin v ja $\top$ kuuluvat. Näin saatu lauserelaatio annetaan argumentiksi k-saturaatioalgoritmille. Rivillä 15 yhdistetään haarat ottamalla R:ään mukaan ne assosiaatiot, jotka ovat mukana sekä R_1:ssä että R_2:ssa. Huomattakoon, että relaatio R ei voi pienentyä tässä operaatiossa.

4.3 Todistusalgoritmi

Varsinainen todistusalgoritmi tekee oletuksen, että F evaluoituu epätodeksi jollakin muuttujien arvonjakelulla, ja kutsuu saturaatioalgoritmia kasvavilla k:n arvoilla aloittaen 0-saturaatiosta.

```
   prove(formula F) :=

 1 T := convert F to triplets
 2 f := the triplet variable representing entire F
 3 R := {f/FALSE}
 4 k := 0
 5 while (true)
 6   R := saturate(T, R, k)
 7   if (R == "contradiction")
 8     exit("tautology")
 9   end if
10   k := k + 1
11 end while
```

Funktio ei jää pyörimään ikuiseen silmukkaan, sillä viimeistään kun k on sama kuin muuttujien lukumäärä, kaikki haarat päätyvät ristiriitaan ja todistus päättyy rivillä 8, tai sitten 0-saturaatio on löytänyt vastamallin ja keskeyttänyt algoritmin `exit`-komennolla.

Algoritmi ajaa k-saturaation loppuun ennen kuin yrittää $k + 1$ -saturaatiota. Tämä on viisasta kahdesta syystä. Ensinnäkin on vaikea sanoa etukäteen, mikä on annetun lauseen vaikeusaste. Siksi kannattaa etsiä ensin lyhyttä todistusta, jos sellainen sattuisi olemaan olemassa. Toiseksi k-saturaatiolla voidaan saada kerättyä huomattavasti lisää informaatiota relaatioon R, vaikka todistusta ei löytyisikään. Tämä helpottaa $k + 1$ -saturaation ajamista. Sama informaatio löydettäisiin toki $k + 1$ -saturaatiollakin, mutta tällöin samaa päättelyä pitäisi suorittaa useassa haarassa ja todistukseen tulisi paljon redundantteja osia.

Algoritmin aikakompleksisuus on $O(|F|^{2k+1})$ [1], missä $|F|$ on F:n muuttujaesiintymien määrä plus konnektiivien määrä ja k on vaikeusaste. Nähdään, että jos vaikeusaste on 0, ajoaika on suoraan verrannollinen lauseen pituuteen. Kiinteällä k:lla ajoaika on polynominen.

Ajoaika riippuu paljon voimakkaammin ongelman vaikeusasteesta kuin lauseen pituudesta. Pitkän lauseen, jonka vaikeusaste on vähintään 2, todistaminen on usein käytännössä liian vaikeaa [2]. Vaikeusastetta ei voida triviaalisti päätellä lauseen rakenteesta. Esimerkiksi pätevän lauseen

$$(a \wedge (b \vee c)) \rightarrow ((a \wedge b) \vee (a \wedge c))$$

vaikeusaste on 0, mutta jos implikaation kääntää oikealta vasemmalle, vaikeusaste kasvaakin 1:een. Yleisesti voidaan sanoa, että vaikeusaste on maksimilukumäärä toisistaan riippumattomia oletuksia, jotka on tehtävä todistusta etsittäessä.

Kun algoritmia on sovellettu verifikaatio-ongelmiin teollisuudessa, on havaittu, että mitä selkeämpi struktuuri verifioitavalla järjestelmällä on, sitä alempi on verifioinnin vaikeusaste. Tämän perusteella voisi kuvitella, että Stålmarckin menetelmä ei ole parhaimmillaan satunnaisesti generoitujen lauseiden tarkastuksessa, sillä satunnaisuus voi aiheuttaa paljon omituisia riippuvuuksia muuttujien välille. Algoritmi onkin pärjännyt paremmin käytännön sovelluksissa kuin menetelmien välisissä kilpailuissa [1].

Harrisonin implementaation [2] perusteella algoritmin tilavaativuus näyttäisi olevan $O(|F|)$. Ilmeisesti ajoaika muodostuu ongelmaksi ennemmin kuin tarvittava muistin määrä.

4.4 Todistuksen redundanssista

Alkeellisissa todistusmenetelmissä käy helposti niin, että hakuavaruus sisältää useita osia, joissa oleellisesti todistetaan samaa asiaa. Saturaatioalgoritmin käyttäminen rajoittaa paitsi turhaa haarautumista, myös tällaista redundanssia. Tutkitaan tästä esimerkkinä konjunktiivisessa normaalimuodossa olevan lauseen

$$F = (a \vee b) \wedge (\neg a \vee \neg b) \wedge (c \vee d) \wedge (c \vee \neg d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$$

toteutumattomuuden tarkastamista naiivilla DPLL-tyyppisellä algoritmilla ja toisaalta Stålmarckin menetelmällä.

Olkoon DPLL-menetelmässämme käytössä ainoastaan yksikköresoluutiosääntö, komplementti puuttuu -sääntö ja haarautumissääntö. Näistä voidaan aluksi soveltaa ainoastaan haarautumissääntöä. Valitaan (lyhytnäköisesti) haarautumismuuttujaksi a.

Tutkitaan haaraa, jonka oletuksena on $a \equiv \top$. Kahdesta ensimmäisestä klausuulista jää jäljelle ainoastaan klausuuli $(\neg b)$, joten yksikköresoluution mukaan b:n on oltava epätosi. Neljä viimeistä klausuulia säilyvät muuttumattomina.

Haaraudutaan uudestaan muuttujan c suhteen. Oletus $c \equiv \top$ tuottaa klausuulit (d) ja $(\neg d)$, mikä havaitaan ristiriidaksi yksikköresoluutiolla. Oletuksella $c \equiv \bot$ päädytään samaan. Siis alkuperäinen oletus $a \equiv \top$ oli väärä.

Seuraavaksi kokeillaan haaraa $a \equiv \bot$. Täällä b:n on oltava tosi (yksikköresoluutio), mutta c:n suhteen tehdään täsmälleen samanlainen päättely kuin viimeksikin.

Todistuksen ongelma on se, että a:n arvo ei mitenkään vaikuta päättelyyn, jolla neljä viimeistä klausuulia saadaan ristiriitaisiksi. Todellinen DPLL-algoritmin toteutus havaitsisi tämän ja käyttäisi jonkinlaista peruutustekniikkaa karsiakseen turhan haaran pois hakuavaruudesta. Toinen vaihtoehto olisi käyttää parempaa hakuheuristiikkaa ja haarautua ensisijaisesti c:n tai d:n suhteen.

Tutkitaan, miten toteutumattomuus todistettaisiin Stålmarckin menetelmällä, johon ei myöskään ole implementoitu peruutusta tai hakuheuristiikkaa. Hajotetaan F ensin alilauseiksi

$$(((((a \overset{e}{\vee} b) \overset{f}{\wedge} (\neg a \overset{g}{\vee} \neg b)) \overset{h}{\wedge} (c \overset{i}{\vee} d)) \overset{j}{\wedge} (c \overset{k}{\vee} \neg d)) \overset{l}{\wedge} (\neg c \overset{m}{\vee} d)) \overset{n}{\wedge} (\neg c \overset{o}{\vee} \neg d)$$

ja muodostetaan kolmikot

$$\begin{array}{rlll} \neg e \equiv \neg a \wedge \neg b, & f \equiv e \wedge g, & \neg g \equiv a \wedge b, & h \equiv f \wedge i, \\ \neg i \equiv \neg c \wedge \neg d, & j \equiv h \wedge k, & \neg k \equiv \neg c \wedge d, & l \equiv j \wedge m, \\ \neg m \equiv c \wedge \neg d, & n \equiv l \wedge o, & \neg o \equiv c \wedge d. & \end{array}$$

Tehdään vastaoletus "F on toteutuva" (tai "$\neg F$ ei ole pätevä") asettamalla $n/\top$ ja aloitetaan 0-saturaatio. Soveltamalla toistuvasti sääntöä, jonka premissi on $\top \equiv y \wedge z$, saadaan $l/\top$, $o/\top$, $j/\top$, $m/\top$, $h/\top$, $k/\top$, $f/\top$, $i/\top$, $e/\top$, $g/\top$. Kun tehdään sijoitukset, jäljelle jäävät kolmikot ovat

$$\begin{array}{llll} \bot \equiv \neg a \wedge \neg b, & \bot \equiv a \wedge b, & \bot \equiv \neg c \wedge \neg d, & \bot \equiv \neg c \wedge d, \\ \bot \equiv c \wedge \neg d, & \bot \equiv c \wedge d. & & \end{array}$$

Näihin ei enää voi soveltaa yksinkertaisia sääntöjä, joten 0-saturaatio on valmis. Aloitetaan 1-saturaatio haarautumalla muuttujan a suhteen. Haarassa $a/\top$ kaksi ensimmäistä kolmikkoa saavat muodon $\bot \equiv \bot \wedge \neg b$, $\bot \equiv \top \wedge b$. Näistä ensimmäinen ei sisällä informaatiota, mutta toiseen voi soveltaa $x \equiv \top \wedge z$ -sääntöä, jonka mukaan x/z eli tässä tapauksessa saadaan assosiaatio $\bot/b$.

Haarassa $a/\bot$ saadaan vastaavasti $\bot/\neg b$. Kun haarat yhdistetään, nähdään että molemmissa tapauksissa a:lla on vastakkainen totuusarvo kuin b:llä, joten saadaan assosiaatio $a/\neg b$. Tämä tekee kaksi ensimmäistä kolmikkoa ja samalla muuttujat a ja b tarpeettomiksi.

Tehdään toinen haarautuminen muuttujan c suhteen. Valinnalla $c/\top$ kaksi viimeistä kolmikkoa tulevat muotoon $\bot \equiv \top \wedge \neg d$, $\bot \equiv \top \wedge d$. Ensimmäisestä saadaan $\bot/\neg d$ ja jälkimmäisestä $\bot/d$, eli päädytään ristiriitaan. Vastaavasti jos otetaan $c/\bot$, joudutaan ristiriitaan kahden keskimmäisen kolmikon nojalla.

Näin 1-saturaatiolla saatiin todistettua alkuperäinen lause toteutumattomaksi, joten ongelman vaikeusaste oli 1. Todistuksessa ei tarvinnut tehdä samoja päätelmiä useita kertoja. Itse asiassa jokaiseen kolmikkoon sovellettiin yksinkertaista sääntöä täsmälleen yhden kerran, jos mukaan ei lasketa niitä kertoja, kun kolmikko heitetään tarpeettomana pois.

5 Sovelluksia

Stålmarckin menetelmä on patentoitu ja se on käytössä mm. Prover Technology AB:n verifiointityökalussa [1]. Kyseessä on itse asiassa menetelmän laajennettu versio, jossa voi hyödyntää rajoitettua kokonaislukuaritmetiikkaa. Menetelmää on laajennettu muillakin tavoilla, esimerkiksi LTL-logiikkaan.

Menetelmää on käytetty sekä laitteiston että ohjelmiston verifiointiin mm. Saab Military Aircraftilla (hävittäjien laskutelinejärjestelmät) ja Adtranzilla (junien ohjausohjelmistot). Lupaavia kokeita on tehty myös logiikkapiirien verifioinnin parissa. Tällä alueella ongelmat on perinteisesti ratkaistu binääristen päätösdiagrammien (BDD) avulla.

Käytännön sovelluksissa ongelman vaikeusaste on usein 0 tai 1. Tällöin on onnistuttu todistamaan lauseita, joissa on jopa puoli miljoonaa konnektiivia. Adtranzilla on päädytty jopa siihen, että ohjelmoijien on noudatettava tiettyjä periaatteita, jotta ohjelmiston verifioinnin vaikeusaste ei nousisi suuremmaksi kuin 1.

6 Yhteenveto

Stålmarckin todistusmenetelmä on teoreettisesti mielenkiintoinen ja käytännössä tehokkaaksi havaittu tapa ratkaista lauselogiikan pätevyysongelma. Sen toimivuus perustuu kahteen asiaan:

- Yksinkertaisten sääntöjen avulla saadaan selville merkittäviä totuuksia vähällä vaivalla.
- Dilemma-säännön ansiosta todistus sisältää sekä rinnakkaisia että peräkkäisiä osia. Näin voidaan välttää turhaa toistoa ja ylimääräistä haarautumista.

Todistuksen löytymisen kannalta olennaista on, mikä on lauseen vaikeusaste.

DPLL-menetelmiin verrattuna menetelmän vahvuus on siinä, että päättelysäännöillä ei saada ainoastaan tietoa siitä, onko jokin muuttuja tosi vai epätosi, vaan myös eri muuttujien välisiä assosiaatioita, kuten että kahdella muuttujalla on sama totuusarvo.

Viitteet

[1] Mary Sheeran, Gunnar Stålmarck, "A Tutorial on Stålmarck's Proof Procedure for Propositional Logic", Formal Methods in System Design 16:1, Jan 2000.

[2] John Harrison, "Stålmarck's Algorithm as a HOL Derived Rule", Theorem Proving in Higher Order Logics, Lecture Notes in Computer Science Vol. 1125 (Springer-Verlag, 1996), pp. 221–234.

[3] Martin Davis, George Logemann, Donald W. Loveland, "A machine program for theorem proving", Communications of the ACM 5, 394–397 (1962).