Matemaattiset todistusohjelmat ja niiden sovellukset

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Tammikuu 2026
Aapeli Syren

TURUN YLIOPISTO
Tietotekniikan laitos

Aapeli Syren: Matemaattiset todistusohjelmat ja niiden sovellukset

LuK-tutkielma, 25 s.
Tietojenkäsittelytiede
Tammikuu 2026

Työssä tutkitaan, mitä erilaisia matemaattisten todistusten generointi- ja käsittelyohjelmia ja -menetelmiä on. Tutkimus on toteutettu kirjallisuukatsauksena.
Ensin esitellään ATP-menetelmiä (Automated Theorem Prover), eli automaattisia matemaattisia todistusmenetelmiä. Käsitellään SAT- ja SMT-ratkaisualgoritmeja, resoluutiota, uudelleenkirjoitusta sekä superpositiokalkyyliä, luonnollista päättelyä, induktiota, korkeamman kertaluvun todistusmenetelmiä ja tekoälymenetelmiä. Perehdytään interaktiivisiin todistusohjelmiin eli todistusassistentteihin. Lisäksi tutustutaan todistusohjelmien sovelluksiin. Tutkimuksessa yritetään kartoittaa, mitä ohjelmallisia todistusmenetelmiä on olemassa.
Matemaattisilla todistuohjelmilla ei ole ollut vielä melkein yhtään vaikutusta matematiikkaan. Sen sijaan formaalit menetelmät (engl. Formal methods), joka tarkoittaa ohjelmistojen ja laitteistojen toiminnan matemaattista varmistamista, on suosittu tieteenala.

Asiasanat: Automaattinen todistusohjelma, Interaktiivinen todistusohjelma, Todistusassistentti, Formaalit menetelmät

Sisällys

1 Johdanto

Matematiikassa tutkitaan yleisiä asioiden yhteyksiä sekä erilaisia mielekkäitä ja hyödyllisiä rakenteita. Matematiikan tärkeys on kasvanut teknologian kehittyessä, sillä teknologian luominen vaatii matemaattista tarkkuutta ja ajattelutapaa. Toisaalta matemaatikkojen työ pohjautuu erilaisten työkalujen käyttöön: paperiin, kynään, viivottimeen, laskimeen ja viimeisimpänä lisäyksenä todistusohjelmiin. Tällä tavoin teknologia edesauttaa matematiikan tutkimusta, kun samalla matematiikan tutkimus antaa teoreettisen pohjan tuon samaisen teknologian kehittämiselle.

Teknologia edistää matematiikkaa useilla eri tavoilla. Ensinnäkin se voi varmistaa matemaattisen päätelmän oikeellisuuden ja luotettavuuden. Toiseksi se voi antaa uuden näkökulman matemaattisiin ongelmiin, mikä voi tukea matematiikan oppimista. Kolmanneksi se voi itsenäisesti tehdä laskelmia ja päätelmiä, jotka ovat ihmisille virhealttiita sekä aikaavieviä.

Toisaalta matematiikkaa voidaan hyödyntää varmistamaan teknologian toimivuus. Nykyään on olemassa paljon kriittisiä järjestelmiä, joiden toiminta pitää jopa matemaattisesti todistaa oikeelliseksi, jotta voidaan estää vahinkoja tapahtumasta. [1] [2, s. 1] Matematiikan ja tietojenkäsittelytieteen välillä on siis tarpeellista vuorovaikutusta, ja matemaattiset todistusohjelmat yhdistävät näitä tieteenaloja. Tässä tutkielmassa perehdytään matemaattisiin todistusohjelmiin sekä -menetelmiin ja kartoitetaan niiden ratkaisemia ongelmia.

Tutkielma on tehty kirjallisuuskatsauksena. Tiedonhaku tehtiin käyttämällä tietokantoja Web of Science, Utuvolter ja Arxiv hakemalla otsikoista hakulauseella *"proof verif* OR proof checker OR proof assistant OR (automated theorem prov* OR (ATP AND (proof OR prove* OR verif*))) OR automated reasoning OR interactive theorem prov* OR theorem prover"*. Utuvolterista löytyi tuhansia tuloksia, joten sieltä haettiin edellä mainitulla hakulauseella ensimmäiset sata tulosta, minkä lisäksi haettiin yleensä ensimmäiset sata tulosta hakusanalla *"automated theorem prov*"* hakusanojen *"SAT"*, *"SMT"*, *"resolution"*, *"natural deduction"*, *"induction"*, *"rewrit*"*, *"superposition"*, *"saturation"*, *"HOL"*, *"higher-order logic"* ja *"type theory"* kanssa, hakusanalla *"theorem prov*"* hakusanojen *"learn*"* ja *"neural"* kanssa sekä hakulauseella *"interactive theorem prov* OR proof assistant"*. Lisäksi haettiin hakulauseella *"formal methods OR formal* verif*"* tietokannasta Web of Science ensimmäiset 200 ja Utuvolterista ensimmäiset 600. Lisäksi utuvolterista haettiin taustaan lähteitä hakusanoilla *"introduction to type theory"*, *"introduction to set theory"* ja *"introduction to mathematical logic"*.

Tutkielmassa vastataan seuraaviin kysymyksiin:

TK1 Minkälaisia ohjelmallisia menetelmiä käytetään todistusten tuottamisessa ja käsittelyssä?

TK2 Mitä käytännöllisiä sovelluksia menetelmillä on?

Luvussa kaksi perehdytään matemaattisiin todistuksiin ja eri todistusjärjestelmiin. Luvussa kolme tarkastellaan eri todistusmenetelmiä ja todistusohjelmien toimintaa. Luvussa neljä käsitellään todistusohjelmien käyttökohteita.

2 Tausta

Tässä luvussa vastataan kysymykseen, mikä on matemaattinen todistus. Esitellään ja kategorisoidaan eri todistusohjelmatyypit kolmeen kategoriaan. Kerrotaan eri matemaattisesta järjestelmistä, propositiologiikasta, ensimmäisen kertaluvun logiikasta ja korkeamman kertaluvun logiikasta.

2.1 Matemaattiset todistukset

Yleensä matematiikkaa tutkitaan abstraktilla tasolla ja useiden välivaiheiden yli hypätään [3]. Kuitenkaan tietokoneet eivät ainakaan vielä pysty ymmärtämään liitutauluille hahmoteltuja kuvioita sellaisenaan, vaan matematiikka pitää kirjoittaa koneluettavaan muotoon. Tämä tapahtuu käyttäen *formaalia kieltä* [3] tai *formaalia systeemiä* [4].

Formaaleissa systeemeissä on alkutiloja sekä päättelysäännöt, joiden avulla voidaan päästä aikaisemmista tiloista uusiin tiloihin. Näitä kutsutaan yhdessä *aksioomiksi*. Tiloja, joihin on päästy alkutiloista käyttämällä muutossääntöjä oikeellisesti kutsutaan *teoreemoiksi*.

Tässä on eräs primitiivinen systeemi [1]:

```
Alkutila: A

Päättelysääntö 1: A -> Ab

Päättelysääntö 2: A -> a
```

[1] Erityisesti *kontekstivapaa kielioppi*

Tässä systeemissä voidaan muodostaa eli *todistaa* sellaisia *kaavoja* kuin “a“, “ab“, “abb“, “abbb“ ja niin edespäin. Kaavan “ab“ todistus näyttää tältä:

```
1. A    (Alkutila)
2. Ab   (Päättelysääntöä 1 sovellettiin riviin 1.)
3. ab   (Päättelysääntöä 2 sovellettiin riviin 2.)
```

Tällainen suoraviivainen formaali matemaattinen todistus on selvästi koneen käsiteltävissä.

2.2 Todistusten ohjelmallinen tuottaminen

Matemaattisten todistusten ohjelmallinen tuottaminen voi tarkoittaa useampaa eri asiaa. Tässä tunnistetaan kolme eri ohjelmatyyppiä: tarkastusohjelma, assistenttiohjelma ja automaattinen todistusohjelma. Termi, joka käsittää kaikki todistusten tuottamistavat ja niiden sovellukset, on *automaattinen päättely* (engl. automated reasoning).

1. Todistustarkastaja tarkistaa, onko matemaattisessa todistuksessa virheitä. [3]

2. Todistusassistentti on ohjelma, joka avustaa ihmistä tuottamaan todistuksia. Usein eri todistuksissa on samanlaisia elementtejä, joita assistentti pystyy tuottamaan automatisoiden todistusprosessia. Toinen paljolti synonyyminä termin todistusassistentti kanssa käytetty termi on ITP (engl. Interactive Theorem Prover) eli vuorovaikutteinen todistusohjelma. [1]

3. Automaattinen todistusohjelma ATP (engl. Automated Theorem Prover) pystyy tuottamaan kokonaisia todistuksia itsenäisesti. [1]

Seuraavaksi esitellään joitakin tärkeitä matemaattisia formaaleja systeemejä, joiden tutkimiseen monet todistusohjelmat perustuvat.

2.3 Propositiologiikka ja ensimmäisen kertaluvun logiikka

Eräs matematiikan alkeellisimmista formaaleista systeemeistä on *propositiologiikka*. Siinä on kahdenlaisia objekteja, propositiomuuttujia ja konnektiiveja. Propositiomuuttujat voivat saada arvon tosi tai epätosi. Konnektiiveja ovat esimerkiksi "ja" ($\wedge$), "tai" ($\vee$), "ei" ($\neg$) ja "jos"($\rightarrow$). [5, s. 115] Propositiomuuttujia yhdistetään konnektiiveilla ja saadaan *kaavoja*, joita voidaan myös joissain tilanteissa kutsua lauseiksi. Kaavoille määritellään totuusarvo. [5, s. 115] Suomen kielen lause "Jos sataa, niin otan sateenvarjon" lausuttaisiin propositiologiikassa näin:

```
sataa → otan sateenvarjon
```

Jokainen propositiologiikan kaava voidaan muuttaa *konjunktiiviseen normaalimuotoon* (engl. Conjunctive normal form, CNF) eli muotoon $(A_1 \vee A_2 \vee ...) \wedge (B_1 \vee B_2 \vee ...) \wedge ...$ missä A_n, B_n ja niin edespäin ovat *literaaleja* eli yksittäisiä propositiomuuttujia tai niiden negaatioita. [6, s. 53] [7, s. 6] Konjunktiivisen normaalimuodon voi myös kuvata joukkona $\{\{A_1, A_2, ...\}, \{B_1, B_2, ...\}, ...\}$. *Klausuuleja* $(P_1 \vee P_2 \vee ...)$ (tai $\{P_1, P_2, ...\}$) kutsutaan *disjunktioiksi*. [5, s. 120]

SAT eli lauselogiikan toteutuvuusongelma on ongelma, jossa tarkastellaan, onko tietty kaava toteutuva, eli voiko kaava saada joillain propositiomuuttujien arvoilla arvon tosi. [7, s. 6] Sateenvarjoesimerkki on toteutuva, koska jos määritellään lauseet "sataa" ja "otan sateenvarjon" todeksi, niin lause "sataa $\rightarrow$ otan sateenvarjon" on tosi. Totuusarvon määrittelemiseen ei tässä tutkielmassa perehdytä. SAT:n ratkaiseminen on NP-vaikea [7, s. 1], joten nykyään ei tiedetä onko olemassa polynomisessa ajassa toimivaa SAT:n ratkaisualgoritmia. On kuitenkin algoritmeja, jotka toimivat tietyissä tilanteissa polynomiajassa. [5, s. 116]

Propositiologiikkaa voidaan laajentaa käsittelemään myös kaavojen sisäistä rakennetta. Esimerkiksi voidaan lisätä verbejä, adjektiiveja ja nomineja eli matemaat-

tisesti sanottuna *predikaatteja*, *funktioita* ja *yksilöitä*. [5, s. 118] Yksilöt ovat *termejä*, ja funktiot ottavat syötteekseen ja palauttavat termejä. [5, s. 119] Predikaatit ottavat syötteekseen termejä ja täytetyt predikaatit ovat propositioita. [5, s. 119] Näihin rakennuspalikoihin perustuu *ensimmäisen kertaluvun logiikka* (engl. First order logic, FOL). [5, s. 118] Ensimmäisen kertaluvun logiikkaa kutsutaan myös *predikaattilogiikaksi* [8, s. 6]. Tässä on aiempi sateenvarjoesimerkki ensimmäisen kertaluvun logiikassa:

```
sataa → otan(sateenvarjo)
```

Yllä olevassa lauseessa esiintyy predikaatti "otan" joka saa syötteekseen yksilön "sateenvarjo".

Lisäksi ensimmäisen kertaluvun logiikassa on yksilömuuttujia, kuten x, ja operaattorit, täsmällisemmin *kvanttorit*, $\forall$ ja $\exists$ [5, s. 118], jotka vastaavat suomenkielen sanoja "jokaiselle" ja "on olemassa". Esimerkiksi lause "$\exists x(otan(x))$" tarkoittaa "On olemassa x, jonka otan" ja "$\forall x(otan(x))$" tarkoittaa "Otan jokaisen x:n". Operaattorit $\forall$ ja $\exists$ siis ottavat parametrikseen muuttujan sekä tuon muuttujan mahdollisesti sisältävän kaavan ja palauttavat totuusarvon.

Ensimmäisen kertaluvun logiikka voi ilmaista suuren osan tutkitusta matematiikasta. Suurin osa matemaattisesta tutkimuksesta joko tehdään tai voitaisiin tehdä etenkin ensimmäisen (tai suuremmankin) kertaluvun *joukko-opissa*. [6] [9, s. 59] [10, s. 65] Joukko-opissa tarkastellaan joukkoja eli asioiden kokoelmia.

SMT-ongelma (engl. Satisfiability modulo theories) on eräs SAT:n useista yleistetyistä versioista ensimmäisen kertaluvun logiikassa. Se laajentaa SAT:tä muun muassa lisäämällä siihen yhtäsuuruuspohjaisen päättelyn, aritmetiikan, bittivektorit ja taulukot. [11, s. 110] [12, s. 221]

2.4 Korkeamman kertaluvun logiikka, tyyppiteoria ja todistusten ja ohjelmien samankaltaisuus

Luonnollisesti ensimmäisen kertaluvun logiikkaa voidaan edelleen laajentaa *korkeamman kertaluvun logiikaksi* (engl. Higher order logic, HOL). Siinä funktiot ja predikaatit voivat ottaa syötteekseen sekä palauttaa mielivaltaisia funktioita. [13, s. 201] Lisäksi näitä kaikkia voidaan kvantifoida kvanttoreilla. [1, s. 4]

Kuitenkin lienee syytä rajoittaa funktioita ja predikaatteja niin, etteivät ne voi ottaa syötteekseen itseään tai palauttaa itseään, tai muuten törmättäisiin ristiriitoihin. [14, s. 17] Tämä lievä rajoitus saadaan aikaiseksi *tyyppiteoriassa*. Tyyppiteoriassa funktioille määritellään tyyppi siten, että funktioilla on pienemmän asteen palautusarvoja sekä argumentteja. [13, s. 201] Edellisen sateenvarjoesimerkin predikaatti tai toisin sanoen totuusarvoinen funktio `otan` on tyyppiä `yksilö → totuusarvo`.

Lisäksi on olemassa korkeamman tason tulkinta tyyppiteoriasta, jossa funktioiden sijaan keskitytään niiden tyyppeihin, jotka voidaan tulkita kaavoina. Tämän tulkinnan nimi on Curry–Howard-vastaavuus. Se on yhtäläisyys matemaattisten todistusten ja tietokoneohjelmien välillä. Sen perusteella jokainen matemaattinen ongelma voidaan muuttaa tyypiksi, ja sen ratkaiseminen vaatii, että löydetään sen tyyppinen funktio, joka on siis joku ohjelma. Tällä tavalla matemaattinen todistus *on* tietokoneohjelma tietyn tulkinnan perusteella, ja samoin tietokoneohjelma *on* matemaattinen todistus, ohjelmointi *on* todistamista [3].

Esimerkiksi identiteettifunktiolla `id` on geneerinen tyyppi `a → a`. Se voidaan tulkita propositioksi "Jos a, niin a", mikä on ilmiselvästi aina totta. Jos sataa, niin sataa. Jos voimme löytää identiteettifunktiolle laillisen määritelmän, kuten `id(x) = x`, niin olemme todistaneet kaavan `a → a`.

Useat todistusohjelmat perustuvat funktionaaliseen ohjelmointiin. Se on luonnollinen valinta, sillä se vastaa korkeamman kertaluvun logiikkaa, ja sitä voi myös

laajentaa Curry–Howard-vastaavuuden mukaan kattamaan niin sanotut *riippuvaiset tyypit* (engl. dependent types), joiden määritelmä riippuu elementaarisesta arvosta. [15, s. 2] Esimerkiksi $n : N \vdash Nat(n) : Type$ on tyyppi, joka sisältää kaikki luvut ykkösestä n:ään. [16, s. 584]

Tyyppiteoria on hyvä valinta tietojenkäsittelytieteeseen. [13, s. xii]. Toisaalta joukko-oppi saattaa soveltua paremmin matemaatikoille. Tyyppiteoria voi soveltua joukko-oppia paremmin komputaation mallintamiseen. [17, s. 1151]

3 Todistusohjelmat

Usein todistusohjelmat ovat suuria yhdistelmiä pienemmistä todistusalgoritmeista ja vaativat paljon laskentatehoa. [18, s. 62] Tässä luvussa esitellään ensin joitain ATP:den monista todistusmenetelmistä. Sitten tarkastellaan ITP:den toimintaa.

3.1 Automaattisten todistusohjelmien toiminta

Automaattinen todistusten generointi on yksinkertaisimmillaan sitä, että annetaan ohjelmalle aksioomat ja laitetaan se yhdistelemään aksioomista uusia teoreemoja. Kuitenkin tämä menetelmä on kuin etsisi neulaa heinäsuovasta, joten usein ohjelmalle annetaan *heuristiikkoja*, jotka päättävät mitä aksioomia on hyvä käyttää ja missä tilanteissa. [4, s. 161] Usein mielekkäiden teoreemojen todistamisessa ATP:tä käyttäen laitetaan se todistamaan pienempiä *lemmoja* eli välitodistuksia, joiden pohjalta todistetaan (tai osoitetaan vääräksi) alkuperäinen teoreema.

SAT ja SMT-ratkaisimia on monenlaisia. Holden (2021) esittelee kolme kategoriaa SAT-ratkaisimille: [7, s. 7]

1. Paikalliseen hakuun perustuvat ratkaisimet
2. Ratkaisimet, jotka eivät perustu paikalliseen hakuun
3. *Portfolio*-ratkaisimet, jotka käyttävät useita eri ratkaisimia

Paikallisen haun algoritmi on kuvattu Ohjelmalistaus 1:ssä. Paikallisen haun algoritmit eroavat toisistaan pääasiassa siinä, miten ne valitsevat muuttujan jokaisella

Ohjelmalistaus 1 Paikallisen haun algoritmi

```
1: procedure PAIKALLINEN HAKU(φ)
2:     valitse (tyypillisesti satunnaiset) alkuarvot x propositiomuuttujille φ:ssä
3:     käännöt = 0
4:     while käännöt < kääntöYläraja do
5:         if x toteuttaa φ:n then
6:             return x
7:         käännöt = käännöt + 1
8:         valitse muuttuja x_i, jonka totuusarvo käännetään
9:         käännä x_i:n totuusarvo x:ssä
```

[7, s. 8]

iteraatiolla. [7, s. 8] Toinen kategoria on osoittautunut kaikista vaikutusvaltaisimmaksi. Se perustuu *DPLL*-algoritmiin (Davis, Putnam, Logemann, Loveland) [7, s. 7]

Klausuuli on *yksikköklausuuli* jos se koostuu vain yhdestä literaalista. *Yksikköpropagaatiossa* on konjunktiivisessa normaalimuodossa oleva kaava $(a_1 \vee a_2 \vee ...) \wedge (b_1 \vee b_2 \vee ...) \wedge ... \wedge p$, missä p on literaali ja samoin a_n, b_n ja niin edelleen, ja tästä kaavasta poistetaan klausuulit $(p \vee ...)$ ja muunnetaan klausuulit $(q_1 \vee ... \vee \neg p \vee q_n ...)$ muotoon $(q_1 \vee ... \vee q_n ...)$, eli poistetaan klausuulista $\neg p$. Kaava sisältää literaalin l puhtaasti, jos se sisältää l:n vain muodossa $l = v$ tai vain muodossa $l = \neg v$. *Puhtaan literaalin eliminaatio* toimii niin, että poistetaan kaavasta $(a_1 \vee a_2 \vee ...) \wedge (b_1 \vee b_2 \vee ...) \wedge ...$, joka sisältää l:n puhtaasti, ne klausuulit $(p_1 \vee p_2 \vee ...)$, jotka sisältävät l:n. [7, s. 7]

DPLL-algoritmi toimii Ohjelmalistaus 2:ssä esitetyllä tavalla. Tyhjä disjunktio tarkoittaa tyhjää joukkoa {} CNF:ssä $\{\{a_1, ...\}, ..., \{\}, ...\}$, eli joukkoa jonka yksikköpropagaatio on tyhjentänyt ottamalla klausuulista $\{\neg p\}$ pois $\neg p$:n.

SMT-ratkaisimet ovat variaatioita DPLL-algoritmista erikoisominaisuuksin. [5, s. 121] SMT-ratkaisimet ja muut ATP:t täydentävät toisiaan, SMT-ratkaisimet käsittelevät paremmin kaavoja ilman muuttujia ja muut ATP:t kvanttoreita. [11, s. 110]

SMT-ratkaisualgoritmit voidaan luokitella kahteen kategoriaan, innokkaaseen (engl. eager) ratkaisutapaan ja laiskaan (engl. lazy) ratkaisutapaan. Innokkaassa ta-

Ohjelmalistaus 2 DPLL-algoritmi

```
1: procedure DPLL(φ)
2:     while φ sisältää yksikköklausuulin l do
3:         φ = yksikköpropagoi(l,φ)
4:     while φ sisältää literaalin l puhtaasti do
5:         φ = eliminoi_puhdas_literaali(l,φ)
6:     if φ on tyhjä then
7:         return true
8:     if φ sisältää tyhjän disjunktion then
9:         return false
10:    l = valitse_satunnainen_literaali(φ)
11:    return DPLL(φ ∧ l) or DPLL(φ ∧ ¬l)
```

[7, s. 7]

vassa SMT-ongelma koodataan suoraan SAT-ongelmaksi. Innokkaan tavan hyvänä puolena on se, että siinä voidaan hyödyntää tehokkaita SAT-ratkaisimia. Laiskassa tavassa käytetään sekä SAT-ratkaisinta että muita ratkaisimia [19, s. 2057]. Näitä muita ratkaisimia kutsutaan *teoriatulkeiksi*. Teoriatulkin tehtävänä on päätellä onko sille annettujen literaalien joukko toteutuva kyseessä olevassa teoriassa T. [20, s. 854] Suurin osa huippuluokan ratkaisimista käyttää laiskaa tapaa. [19, s. 2057]

Resoluutio on alla oleva päättelysääntö propositiologiikassa ja ensimmäisen kertaluvun logiikassa [1] :

$$\frac{a_1 \vee a_2 \vee ... \vee c_1,\ b_1 \vee b_2 \vee ... \vee \neg c_2}{a_1\alpha \vee a_2\alpha \vee ... \vee b_1\alpha \vee b_2\alpha \vee ...}$$

missä a_n:t eivät jaa muuttujia b_n:en kanssa, missä c_1 ja c_2 ovat *yhtenäistettävissä kaikkein yleisimmällä yhtenäistäjällä* eli tietyllä korvauksella α ja missä $a_n\alpha$ ja $b_n\alpha$ ovat vastaavasti a_n ja b_n korvauksen α applikaation jälkeen. Yhtenäistettävyys tarkoittaa, että c_1 ja c_2 voidaan saattaa samaan muotoon korvauksella, joka löydetään yhtenäistämisalgoritmilla. [18, s. 66] [5] Esimerkiksi c_1 voisi olla $Q(x_1, ..., x_n, ...)$ ja c_2 voisi olla $Q(x_1, ..., f(y), ...)$, koska ensimmäisessä kaavassa termin x_n voi korvata termillä $f(y)$. Vaakaviivan yläpuolella olevat klausuulit ovat disjunktiomuodossa olevia olettamuksia ja alapuolella oleva klausuuli on todistettava

[1] korkeamman kertaluvun logiikassa sääntö on vaikeampi implementoida

disjunktiomuodossa oleva klausuuli. Resoluutiosääntö tarkoittaa että jos on kaava "a_1 *tai* a_2 *tai ... tai* c_1" ja kaava "b_1 *tai* b_2 *tai ... tai ei* c_2", niin päätellään kaava "$a_1\alpha$ *tai* $a_2\alpha$ *tai ... tai* $b_1\alpha$ *tai* $b_2\alpha$ *tai ...*".

Resoluutiotodistus on todistus kumoamisen kautta, [17, s. 28] eli oletetaan ensin todistettavan kaavan a negaatio $\neg a$, ja sitten yritetään päästä resoluutiosäännöllä tyhjään klausuuliin. Eli koitetaan löytää jokin klausuuli b ja sen negaatio b, mistä seuraa tyhjä klausuuli, mikä tulkitaan ristiidaksi. [4, s. 163]

Resoluutiossa on usein paljon yhdistettäviä klausuuleja, joten on hyvä valita toimiva heuristiikka, joka ohjaa todistuksen kulkua järkevään suuntaan. Eräs heuristiikka on se, että valitaan resoluution lähtöklausuuliksi ainakin yksi klausuuli, joka on literaali. [4, s. 163]

Logiikkaohjelmointikieli Prolog perustuu resoluutioon. [4, s. 163] Sitä on käytetty myös teoreemien todistamiseen. [21, s. 1775] Prolog on herättänyt kiinnostusta resoluutiomenetelmää kohtaan. [18, s. 62]

Uudelleenkirjoitus (engl. rewrite rule) on yksinkertaisuudessaan sääntö $A \to B$, mikä tarkoittaa, että (jossakin kaavassa ϕ) alikaava A voidaan korvata alikaavalla B. [4, s. 167] Esimerkiksi uudelleenkirjoitussääntö voi olla muotoa $max(0, x) \to x$, missä x on luonnollinen luku. [17, s. 538]

Uudelleenkirjoitukselle on vaikea valita heuristiikkoja. Lisäksi uudelleenkirjoitus usein pidentää kaavoja, mikä kuluttaa tietokoneen resursseja, kun taas resoluutio usein lyhentää kaavoja. Kuitenkin uudelleenkirjoitusta on usein intuitiivisempi käyttää kuin resoluutiota. [4, s. 168] Uudelleenkirjoitusta käytetään muun muassa Haskell-kielen kääntäjässä GHC:ssä sekä LLVM:ssä. [22, s. 62]

Superpositiokalkyyli (engl. superposition calculus) on uudelleenkirjoitukseen perustuva päättelyjärjestelmä. [23, s. 477] Se on versio resoluutiokalkyylistä erityisesti ensimmäisen asteen logiikassa yhtäsuuruuden kanssa. [20, s. 860] Superposi-

tiokalkyylin päättelysäännöt ovat jokseenkin monimutkaisia, joten niitä ei esitellä tässä.

Menestyneimmät ja tehokkaimmat ensimmäisen kertaluvun logiikan todistusohjelmat perustuvat *saturaatioon*. [24, s. 1] [25, s. 253] Siinä etsitään todistusta kumoamisen kautta käyttäen yleensä joko resoluutiota tai/ja *superpositiokalkyyliä*, kunnes joko löydetään ristiriita, mikä tarkoittaa että kaava on toteutumaton, saavutetaan *saturoitu* joukko, josta ei voida enää luoda uusia ei-redundantteja teoreemoja, erityisesti ristiriitoja, mikä tarkoittaa että kaava on toteutuva, [25, s. 253] tai kunnes ohjelma käyttää loppuun käytettävissä olevan ajoajan, mikä tarkoittaa että kaavan toteutuvuutta ei saatu selville. [26, s. 13] Saturaatiossa ongelma on ilmaistu joukkona alkuklausuuleja (klausuulimuodossa ilmaistut aksioomat ja tarkasteltavan väitteen negaatio). Saturaatioalgoritmin alkaessa kaikki alkuklausuulit ovat *ei-prosessoidut* -joukossa. Sitten valitaan *annettu klausuuli* tästä joukosta ja siirretään se *prosessoidut*-joukkoon ja muodostetaan kaikki mahdolliset päättelyt prosessoidut-joukon klausuuleista. Uudet muodostetut klausuulit lisätään ei-prosessoitu -joukkoon. Tämän jälkeen valitaan uusi annettu klausuuli ja toistetaan sama prosessi uudelleen ja uudelleen. Tyypillisesti tämä prosessi on kehitetty olemaan *kumouksellisesti täydellinen* (engl. refutationally complete) eli että äärellisessä ajassa ristiriita löytyy, jos todistettavana oleva lause seuraa aksioomista. [27, s. 598, s. 599] Resoluutio on kumouksellisesti täydellinen [28, s. 11] Superpositiokalkyyli on yleisin saturaatioon perustuva ensimmäisen kertaluvun logiikan päättelyjärjestelmä ja johtava ATP-teknologia. [29]

Luonnollinen päättely on todistustapa, jossa käytetään ihmisille intuitiivisia päättelysääntöjä. [4, s. 163] Kuten resoluutiolla, luonnollisella päättelyllä on monia mahdollisia heuristiikkoja ja oikean valinta voi aiheuttaa päänvaivaa. [4, s. 164] Päättelyn "luonnollisuuden" eli ihmisläheisyyden hintana on tehokkaampien mutta ihmisille epäintuitiivisten menetelmien poisjääminen. [30, s. 1156] Toinen ongelma

on se, että luonnollisessa päättelyssä on vaikeaa muodostaa todistus todistettavasta kaavasta taaksepäin aksioomiin. [30, s. 1160] Alla on eräitä luonnollisen päättelyn yleisiä päättelysääntöjä, jotka esittelee Crotts ja Tate (2022): [31, s. 47] :

Modus ponens $\dfrac{\phi \rightarrow \psi,\ \phi}{\psi}$ Modus tollens $\dfrac{\phi \rightarrow \psi,\ \neg\psi}{\neg\phi}$ Disjunction elimination $\dfrac{\phi \vee \psi,\ \neg\phi}{\psi}$

Conjunction elimination 1 $\dfrac{\phi \wedge \psi}{\phi}$

Induktiossa halutaan osoittaa että jokin ominaisuus pätee järjestetylle joukolle alkioita. Ensin todistetaan, että ominaisuus pätee ensimmäiselle alkiolle, ja sitten todistetaan että jos ominaisuus pätee yhdelle jollekin alkiolle, se pätee myös *seuraavalle* alkiolle. Siitä seuraa, että ominaisuus pätee kaikille alkioille. Induktio toimii hyvin rekursiivisesti generoiduille alkiojoukoille. Myös induktiossa usein tarvitaan heuristiikkoja. [4, s. 165] Induktion käyttöä on tutkittu muun muassa SMT-ratkaisimien, resoluution sekä uudelleenkirjoitusjärjestelmien kanssa. [25] Induktio voidaan jakaa kahteen kategoriaan, eksplisiittiseen ja implisiittiseen. Eksplisiittisessä induktiossa induktiosäännöt on eksplisiittisesti sisällytetty todistuksiin. Implisiittisessä induktiossa todistettava väite lisätään aksioomien joukkoon ja sitten ajetaan esimerkiksi Knuth-Bendix täydennysprosessi (joka käyttää uudelleenkirjoitusta), ja jos ristiriitaa ei löydetä, niin väite on todistettu. [17, s. 848]

Korkeamman kertaluvun logiikassa ja tyyppiteoriassa on myös joitain ATP-menetelmiä. Resoluutiota voi käyttää myös korkeamman asteen logiikassa, mutta siihen tarvitaan muun muassa monimutkaisempi yhtenäistämisalgoritmi ja *hajautussääntö*. Toisaalta korkeamman kertaluvun logiikka voidaan ilmaista ensimmäisen kertaluvun teoriana, mutta ensimmäisen kertaluvun resoluutio tässä teoriassa on paljon tehottomampi kuin korkeamman kertaluvun resoluutio. [32, s. 1] Resoluution lisäksi luonnollista päättelyä voi käyttää korkeamman kertaluvun logiikassa, jolloin on käytössä enemmän päättelysääntöjä. [33]

Riippuvaisesti tyypitetylle tyyppiteorialle ei ole melkein yhtään ATP:itä. [33, s. 2] Riippuvaisesti tyypitetyn korkeamman kertaluvun logiikalle on vaikea tehdä ratkaisimia, mutta Rothgang, Rabe ja Benzmüller (2025) ja Rothgang (2023) esittävät tapoja kääntää se tavalliseen korkeamman kertaluvun logiikkaan ja käyttää sen ratkaisimia. [34, s. 1]

Tekoälyyn pohjautuvia menetelmiä on myös jonkin verran. Suuri osa ATP:den implementaatioista on ollut algoritmisia, mutta viime vuosina on alettu sisällyttää koneoppimista ATP:hin [7, s. 1] Viime vuosina syväoppimisen edistyksen ja etenkin suurten kielimallien ilmaantumisen seurauksena on ollut lisääntynyttä tutkimusta näiden tekniikkojen käyttämisessä todistusten tehostamisessa. Itse asiassa tutkimuksien, jotka käsittelevät koneoppimista todistamisessa, määrä on kasvanut kahdesta vuonna 2016 45:een vuonna 2023 [35, s. 1]

Zhaoyu ja muut (2024) jakaa syväoppimismenettelytavat todistamisessa viiteen kategoriaan: Autoformalisaatio, premissinvalinta, todistusaskeleen muodostaminen, todistushaku ja muut. Autoformalisaatiossa pyritään muuntamaan epämuodollinen todistus koneen tarkistettavaan muotoon automaattisesti, mikä on todella vaikeaa. Premissinvalinnassa pyritään löytämään valtavasta joukosta todistettuja lemmoja todistamisessa hyödyllisimmät lemmat. Premissinvalintaa käytetään niin ITP:ssä kuin ATP:ssä. Todistusaskeleen muodostamisessa pyritään ennustamaan yksi tai enemmän askeleista, joita tullaan käyttämään todistuksessa. Todistushaussa pyritään systemaattisesti käymään läpi potentiaalisten todistuspolkujen avaruus, jotta pystytään muodostamaan validi todistuspuu. [35, s. 3, s. 4, s. 5]

Hyvien heuristiikkojen valinta on vaikeaa, mutta koneoppiminen tarjoaa tavan kiertää tämän ongelman ja vähentää vaadittavaa ihmisen tekemää työtä ATP:den käytössä uusissa toimialueissa. [24, s. 1] Esimerkiksi E-ohjelmalla on yli 80 heuristiikkaa. [36, s. 154]

Zhaoyu ja muut (2024) esittelevät joitain syväoppimisen ongelmia todistamisessa: datan niukkuus, todistusohjelmien evaluointi ja ihmis-tekoäly-interaktio. Formaalin todistusdatan määrä on kasvussa, mutta on vielä niukempaa kuin muissa alueissa, missä LLM:t ovat menestyneet. Esimerkiksi todistusohjelma Isabellen Formaalien Todistusten Arkistossa on 250 000 todistusta ja Lean-ohjelman mathlib-kirjastossa on 140 000 todistusta, mikä riittää vielä vain pienemmille malleille. Verrattuna perinteisiin syväoppimisen tehtäviin kuten klassifikaatioon, on paljon haastavampaa evaluoida todistusohjelmien tehokkuutta kokonaisvaltaisesti. Esimerkiksi suuret mallit suoriutuvat paremmin kun aikaa annetaan enemmän, kun taas pienemmät mallit loistavat pienemmillä aikamäärillä. Kolmanneksi tutkimuksista on seurannut melko vähän hyödyllisiä työkaluja ihmismatemaatikkojen avustamiseen. [35, s. 9, s. 10]

ATP-ohjelmia on olemassa monta. Joka vuosi järjestetään CASC-kilpailu (CADE ATP System Competition). Eräitä kilpailuissa menestyneimpiä ATP:itä ovat Satallax, Niptick, Vampire, Beagle, E ja LEO-III. E ja Vampire perustuvat ensimmäisen kertaluvun superpositiokalkyyliin ja Satallax korkeamman kertaluvun logiikkaan. [1, s. 9]

3.2 Interaktiiviset todistusohjelmat

Ongelmat voivat olla joko *kompleksisia* tai *syviä*. Kompleksiset ongelmat ovat suuria, kun taas syvät ongelmat ovat monimutkaisia. Syvien ongelmien kohdalla automaattisen päättelyn työkalut auttavat vain vähän, kun taas todistusassistentit voivat olla hyödyllisempiä. [17, s. 1152]

Todistusassistenttiyhteisö jakautuu pääosin kahteen menestyneeseen ryhmään. Ensimmäinen ryhmä käyttää ilmaisuvoimaisia tyyppiteorioita perustana, ja toinen ryhmä käyttää klassista joukko-oppia käyttäen *yksinkertaista tyypitystä*. [37, s. 237] Kuitenkin Kunčar ja Popescu (2019) esittelevät tavan muuntaa tyypit joukoiksi,

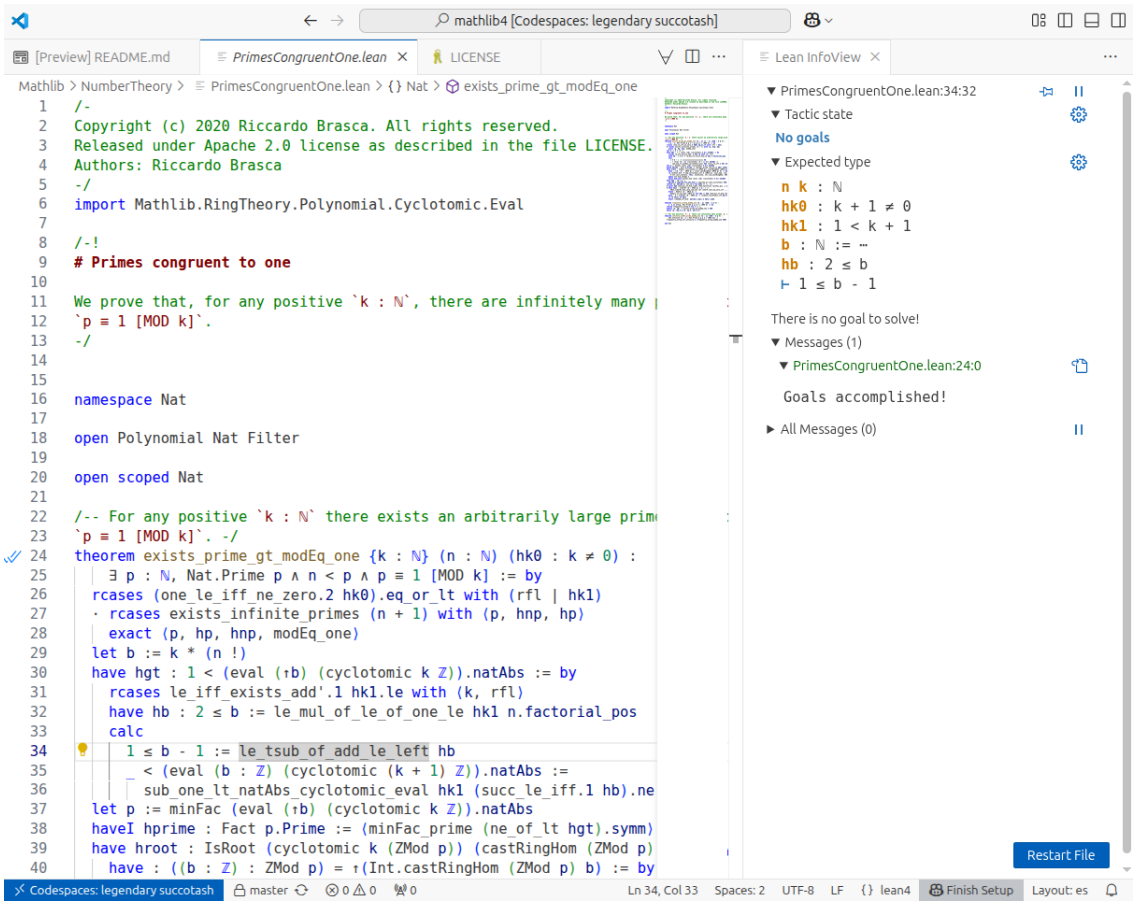
mikä voisi yhdistää näitä ryhmiä. [37, s. 255] Useat interaktiiviset todistusohjelmat käyttävät riippuvaisesti tyypitettyä tyyppiteoriaa. [34, s. 1]

Tyypillisesti todistusassistentit käyttävät korkeamman kertaluvun logiikkaa, kun taas ATP:t käyttävät ensimmäisen kertaluvun logiikkaa. ITP:den ja ATP:den integrointi vaatii korkeamman kertaluvun logiikan kaavojen kääntämistä ensimmäisen kertaluvun logiikkaan. [38, s. 35] ITP:den ja ATP:den yhteistyön mahdollistavat *vasarat* (engl. hammer), jotka toimivat rajapintana niiden välillä [39, s. 257] ja muodostavat todistuksia vain yhdellä klikkauksella [33, s. 1]. Ne mahdollistavat suuriin kirjastoihin pohjautuvan päättelyn [40, s. 102] ja ovat hyödyllisiä suurten todistusten formalisaatiossa [28, s. 49] Vasarat usein koostuvat neljästä osasta, jotka esittelee Jiang (2019):

- Lemmanvalintamoduuli (voi kutsua myös premissinvalintamoduuliksi [40]), joka suodattaa relevantit lemmat, joita ATP:t voivat käyttää
- Kääntäjämoduuli, joka kääntää ITP-ongelman FOL:n syntaksiin, jota ATP:t voivat käsitellä
- Linkkejä ulkoisiin ATP:hin, jotka etsivät sekä tuottavat todistuksia
- Todistusrekonstruktiomoduuli, joka rekonstruoi ATP:n ulostulon vastaavaksi ITP-todistukseksi [41, s. 90]

Yleensä vasaran menestykseen vaikuttavat premissinvalintamoduuli sekä ATP:den määrä, ei niinkään kääntäjä- tai rekonstruktiomoduuli [28, s. 51] Premissinvalinnassa käytetään niin oppimiseen perustumattomia kuin perustuvia menetelmiä ja niiden yhdistelmiä. [40, s. 105] Yleensä sekä vasaralla että ATP:illä on omat premissinvalitsijat, jolloin premissinvalinta tehdään kahdesti. [28, s. 51]

Useimmat todistusassistentit käyttävät *taktiikkoja*. Siinä käyttäjä aloittaa todistettavalla väitteellä ja jakaa väitteen pienempiin, yksinkertaisempiin väitteisiin käyttäen taktiikkoja. [28, s. 15]



Kuva 3.1: Lean-laajennus Visual Studio Code -editorissa

Interaktiivisessa todistamisessa käytetään kolmea todistustyyliä, proseduraalinen, deklaratiivinen ja opastettu automaattinen tyyli. Proseduraalisessa tyylissä todistus muodostetaan taktiikkasekvenssinä. Deklaratiivisessa tyylissä todistukset ovat kontrolloidulla luonnollisella kielellä kirjoitettuja tekstejä. Opastetussa automaattisessa tyylissä käyttäjä syöttää sekvenssin lemmoja, jotka järjestelmä yrittää todistaa itsenäisesti. Opastettua automaattista tyyliä voidaan proseduraalisen tai deklaratiivisen tyylin versiona. [42, s. 1]

Kuvassa 3.1 näkyy Lean-todistusohjelman laajennus Visual Studio Code -editorissa. Oikeassa näkymässä näkyy taktiikan tila.

Eräitä ITP:itä ovat ACL2, Isabelle, Metamath, Twelf, Agda, Mizar, HOL, Nuprl, Rocq ja PVS. Näistä riippuvaisesti tyypitettyjä ovat Twelf, Agda, Mizar, Nuprl, Rocq ja PVS. Vain ACL2 ja Metamath ovat tyypittömiä. ITP:t ovat järjestyksessä luotettavimmasta vähiten luotettavaan Agda, Rocq, Metamath, Nuprl, HOL, Isabelle, Twelf, Mizar, PVS ja ACL2. Suurin osa näistä käyttää funktionaalista ohjelmointitapaa. [1, s. 11, s. 9] [2]

[2]Lisäksi sivustolla https://www.cs.ru.nl/F.Wiedijk/100/index.html esitellään eri todistusohjelmien sadan *tärkeimmän* teoreeman todistusprosentteja.

4 Käyttökohteet

Tässä luvussa tutkitaan todistusohjelmien käyttökohteita. sekä laitteistojen kehittämisessä. Tutkitaan todistusohjelmien käyttökohteita matematiikassa ja niin sanotuissa *formaaleissa menetelmissä*, jotka koskevat laitteistoja ja ohjelmistoja.

Matematiikkaan automaattisella päättelyllä ei vielä ole ollut melkein yhtään vaikutusta. [43, p. 5] Toisaalta ITP:den tärkeys matemaattisessa yhteisössä kasvaa. Kuitenkin on kestänyt, että matemaatikot lämpenevät ajatukselle tietokoneen käyttämisestä täysin formaalissa todistamisessa ja että he luottavat tietokoneiden tuottamiin todistuksiin. [44, s. 5, s. 15] Todistusohjelmien käytön kasvu matemaattisessa tutkimuksessa tulee todennäköisesti olemaan hidasta ja tasaista. Ainakin menneisyydessä todistusohjelmien kirjastojen vaatimattomuus on ollut iso este. [45, s. 11] Riippuvaisesti tyypitetty tyyppiteoria on matemaatikoille parhaiten soveltuva kaikista implementoiduista järjestelmistä. [43, s. 9]

Bundy (2011) esittelee viisi syytä, miksi matemaatikot eivät tahtoisi käyttää automaattisia todistusohjelmia: [45]

1. Formaalit todistukset ovat liian yksityiskohtaisia ja pitkiä.
2. Todistusohjelmat eivät ole tarpeeksi tehokkaita.
3. Todistusohjelmia on tylsää käyttää.
4. Todistusohjelmia on vaikea käyttää.
5. Manuaalinen todistaminen on hauskaa.

Lisäksi hän kertoo, että matemaattisten teorioiden kirjastojen pienuus on este.

Formaalit menetelmät (engl. Formal methods) on tieteenala, jossa pyritään ilmaisemaan laitteistojen sekä ohjelmistojen toiminta matemaattisesti sekä todistamaan, että ne toimivat virheittä. [46, s. 2]

Formaali spesifikaatio tarkoittaa järjestelmän toivottujen ominaisuuksien kuvailemista matemaattisesti ja tarkasti. [47, s. 23] Formaali verifikaatio tarkoittaa matemaattista todistusta siitä, että järjestelmä noudattaa spesifikaatiota. [46, s. 2]

Kaksi suosituinta formaalin verifikaation menetelmää ovat mallintarkastus ja teoreemien todistaminen. [1, s. 2] [46, s. 5] Mallintarkastuksessa ensin muodostetaan äärellinen malli järjestelmästä, ja sitten sen tila-avaruus tutkitaan mallintarkastajalla, jotta pystytään selvittämään toteutuuko jokin haluttu ominaisuus mallissa. [1, s. 2] Mallintarkastus on *kattava* (engl. exhaustive) eli jos jollakin järjestelmän ajolla rikotaan spesifikaatiota, mallintarkastaja tuottaa vastaesimerkin. Mallintarkastajat jakautuvat kahteen kategoriaan, eksplisiittiseen ja symboliseen. Eksplisiittisessä mallintarkastuksessa järjestelmän toiminta-avaruus on esitetty eksplisiittisesti graafina muistissa, ja mallintarkastaja systemaattisesti tutkii jokaisen tilan ja verifioi toteuttaako se spesifikaation. Tämä sopii esimerkiksi pienen tila-avaruuden omaaville järjestelmille. Symbolisessa mallintarkastuksessa esitetään tilojen joukot ja transitioiden joukot symbolisesti käyttäen tietorakenteita, kuten Boolen kaavoja. [46, s. 6] Mallintarkastuksella voidaan esimerkiksi tarkistaa osa järjestelmän spesifikaatiosta, mutta ongelmana on tila-avaruuden suuruus. Toisaalta teoreemien todistamisessa voidaan käsitellä äärettömiä järjestelmiä, jolloin ne määritellään ja spesifioidaan sopivassa matemaattisessa logiikassa ja tärkeät ominaisuudet todistetaan todistusohjelmilla. [1, s. 2]

Vuonna 2025 julkaistussa kirjallisuuskatsauksessa, joka käsitteli tekoälyn soveltamista formaaleihin menetelmiin, selvisi, että käytännön sovelluksia on paljon mutta teoreettista taustaa käsitteleviä tutkimuksia on melko vähän. Tämän lisäksi moni

tutkimus käsitteli teoreemien todistamista ja SAT:n ratkaisua, kun taas melko harva tutkimus käsitteli mallintarkastusta. [48, s. 30]

Ohjelmistoissa formaalit menetelmät voi jakaa kahteen kategoriaan, formaali verifikaatio ja ohjelmasynteesi. Formaalin verifikaation voi jakaa kahteen kategoriaan, kooditason verifikaatio ja abstrakti spesifikaatioverifikaatio. Ohjelmasynteesissä aloitetaan oikeellisuusväitteillä (tyypillisesti ensimmäisen kertaluvun logiikassa), jotka kertovat, missä suhteessa järjestelmän syötteiden ja ulostulon pitäisi olla. [4, s. 169, s. 170]

On arvioitu, että ohjelmiston todistamiseen käytettyjen rivien määrä kasvaa kvadraattisesti verrattuna tarkasteltavan ohjelmiston rivien määrään [49] Esimerkiksi interaktiivisessa todistusohjelmassa Rocq (aiemmin Coq) kirjoitettu ja verifioitu CompCert kääntäjäprojekti vaati 6 henkilövuotta, 175 Rocq-tiedostoa ja 100 000 Rocq koodiriviä. [50, s. 71]

Viime vuosikymmenenä neuroverkot ovat yleistyneet, ja on hyvä varmistaa niiden toimivuus formaalisti. Kuitenkin niiden toiminnan formaali spesifikaatio on vaikeaa ja niitä pidetään *mustina laatikoina*. Silti sen seurauksena, että löydettiin adversariaaliset hyökkäykset neuroverkkoja vastaan, ATP-yhteisö on kehittänyt neuroverkkoverifioijia edeten niinkin nopeasti, että nyt ne pystyvät verifioimaan ominaisuuksia neuroverkoista, joilla on satoja tuhansia solmuja. Eräitä ongelmia ovat se, että ITP:t pystyvät käsittelemään vain pieniä verkostoja verrattuna moderneihin neuroverkkoihin, huollettavuus eli verifikaation päivittäminen neuroverkon päivittyessä ja suorituskyky. Vaikka neuroverkkojen verifikaatiolla on kirkas tulevaisuus, on vielä paljon työtä tehtävänä. [51, s. 1, s. 2, s. 3, s. 15]

5 Pohdintaa

Taulukossa 5.1 esitellään käytetyt lähteet sekä käsittelivätkö ne automaattista todistamista, interaktiivista todistamista tai sovelluksia. Taulukosta huomataan, että aineistoa löytyi tasaisesti eri vuosilta. Teoriaan ja taustaan keskittyviä lähteitä oli melko helppo löytää. Yksittäisiä ATP-menetelmiä löytyi hajanaisesti, ja harva lähde käsitteli asiaa kokonaisvaltaisesti, poislukien Russell (1992). Myös ITP:istä löytyi tietoa jokseenkin hajanaisesti. Formaaleista menetelmistä oli helppo löytää lähteitä ja etenkin kohdistuen ohjelmistoihin. Formaaleista menetelmistä laitteistoissa oli vaikeampaa löytää lähteitä.

Vuonna 2022 tehdystä kyselystä selvisi että 39 % Rocq-todistusohjelman käyttäjistä oli 20-29-vuotiaita, 33 % 30-39-vuotiaita ja 15 % 40-49-vuotiaita, [52, s. 8] mikä vihjaa, että todistusohjelmat ovat ehkä luontevampia nuoremmille sukupolville. Samassa tutkimuksessa selvisi, että 46 %:lla käyttäjistä oli tohtorin tutkinto, 31 %:lla maisterin tutkinto ja 18 %:lla kandidaatin tutkinto, [52, s. 9] mikä vihjaa, että usein todistusohjelmia käyttävät korkeammin koulutetut. Selvisi myös, että 54 % käyttäjistä työskenteli korkeakouluissa, 32 oli % opiskelijoita ja 24 % työskenteli yksityisellä sektorilla ei-akateemisesti. [52, s. 10]

Taulukko 5.1: Taulukko lähteistä ja niiden käsittelemistä osa-alueista

	Automaattinen todistaminen	Interaktiivinen todistaminen	Sovellukset
Ter Beek ja muut (2025)			x
Stock ja muut (2025)			x
Rothgang ja muut (2025)	x		
Oates ja muut (2025)		x	
De Almeida Borges ja muut (2025)		x	
Pantsar (2024)	x		
Li ja muut (2024)	x	x	
Kœhler ja muut (2024)	x		
Azam (2024)			
Avigad (2024)			x
Abdulaziz (2024)		x	x
Scott ja muut (2023)	x		
Rothgang (2023)	x		
Dowek (2023)	x		
Chen (2023)	x		
Abdelaziz ja muut (2023)	x		
Piepenbrock (2022)	x		
Daggitt ja muut (2022)			x
Crotts ja Tate (2022)			
Sieg ja Derakhstan (2021)	x		
Kristiansen (2021)	x	x	
Holden (2021)	x		
Hadju ja muut (2021)	x		
Gauthier ja muut (2021)		x	
Mo ja muut (2020)		x	
Echenim ja Peltier (2020)	x		
Nawaz ja muut (2019)	x	x	x
Kunčar ja Popescu (2019)	x	x	
Jiang (2019)	x	x	x
O'Regan (2017)			x
Montserrat Armstrong (2016)	x		
Blanchette ja muut (2016)	x	x	
Plaisted (2014)	x		
Bridge ja muut (2014)	x		
Kovács ja muut (2013)	x		
Blanchette ja muut (2013)	x	x	
Wiedijk (2012)		x	
Bundy (2011)	x	x	x
Biere (2011)	x		
Geuvers (2009)		x	
Meng ja Paulson (2008)		x	
De Moura ja Bjørner (2008)	x		
Wolf (2005)			
Kamareddine ja muut (2004)			
Slagle ja Shankar (2003)			
Barthe ja Coquand (2002)			
Andrews (2002)			
Robinson ja Voronkov (2001)	x	x	
Enderton (2001)			
Jacobs (1999)			
Russell (1992)	x	x	x
Peikert (1987)	x		

6 Yhteenveto

Matematiikkka on aina hyötynyt työkalujen käytöstä. Ajan saatossa nuo työkalut ovat kehittyneet tehokkaammiksi. Tässä tutkielmassa oli tavoitteena kartoittaa matematiikan uusimpia työkaluja eli ATP- ja ITP-menetelmiä ja -ohjelmia ja niiden sovelluksia.

TK1: Löydettiin suuri määrä ohjelmallisia todistusmenetelmiä. ATP-menetelmät olivat SAT- ja SMT-ratkaisimet, resoluutio, uudellenkirjoitus ja superpositiokalkyyli, luonnollinen päättely, induktio, korkeamman kertaluvun menetelmät ja tekoälymenetelmät, ja todistusassistenteissa löydettiin taktiikat sekä vasarat. Todettiin, että jokaisella menetelmällä on omat vahvuutensa.

TK2: Löydettiin kaksi todistusohjelmien sovellusta, matematiikka sekä formaalit menetelmät. Matematiikassa todistusohjelmilla oli vain vähän vaikutusta, kun taas formaaleissa menetelmissä todistusohjelmia käytettiin runsaasti.

Matemaattisten todistusohjelmien käyttöönotto on ollut verkkaista, ja todennäköisesti tulee jatkumaan jonkin aikaa verkkaisena. Silti uskon, että niillä on edessään jokseenkin kirkas tulevaisuus ja että niiden rooli tulee kasvamaan etenkin turvallisuuskriittisissä järjestelmissä, joiden määrä näyttää lisääntyvän.

Lähdeluettelo

[1] M. S. Nawaz, M. Malik, Y. Li, M. Sun ja M. I. U. Lali. "A Survey on Theorem Provers in Formal Methods". arXiv: 1912.03028 [cs], esijulkaistu.

[2] H. Oates, H. Yun ja N. Gurusinghe. "Theorem Provers: One Size Fits All?" arXiv: 2509.15015 [cs], esijulkaistu.

[3] H. Geuvers, "Proof assistants: History, ideas and future", *Sadhana*, vol. 34, nro 1, s. 3–25, helmikuu 2009, ISSN: 0256-2499, 0973-7677. DOI: 10.1007/s12046-009-0001-5.

[4] S. Russell ja T. W. Unisys, "On Automated Theorem Proving", *Annals of the New York Academy of Sciences*, vol. 661, nro 1, s. 160–173, joulukuu 1992, ISSN: 0077-8923, 1749-6632. DOI: 10.1111/j.1749-6632.1992.tb26040.x.

[5] D. A. Plaisted, "Automated theorem proving", *WIREs Cognitive Science*, vol. 5, nro 2, s. 115–128, maaliskuu 2014, ISSN: 1939-5078, 1939-5086. DOI: 10.1002/wcs.1269.

[6] H. B. Enderton, *A Mathematical Introduction to Logic*, 2nd ed. San Diego: Harcourt/Academic Press, 2001, ISBN: 978-0-08-049646-7.

[7] S. B. Holden, "Machine Learning for Automated Theorem Proving: Learning to Solve SAT and QSAT", *Foundations and Trends® in Machine Learning*, vol. 14, nro 6, s. 807–989, 2021, ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000081.

[8] M. Montserrat Armstrong, *Automated theorem proving*, 2016.

[9] R. S. Wolf, *A Tour through Mathematical Logic* (Carus Mathematical Monographs v. 30). Washington, DC: Mathematical Association of America, 2005, 1 s., ISBN: 978-1-61444-028-4.

[10] R. Azam, *Seminar : Introduction to set theory*, 2024.

[11] J. C. Blanchette, S. Böhme ja L. C. Paulson, "Extending Sledgehammer with SMT Solvers", *Journal of Automated Reasoning*, vol. 51, nro 1, s. 109–128, kesäkuu 2013, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-013-9278-5.

[12] J. Scott, A. Niemetz, M. Preiner, S. Nejati ja V. Ganesh, "Algorithm selection for SMT: MachSMT: Machine learning driven algorithm selection for SMT solvers", *International Journal on Software Tools for Technology Transfer*, vol. 25, nro 2, s. 219–239, huhtikuu 2023, ISSN: 1433-2779, 1433-2787. DOI: 10.1007/s10009-023-00696-0.

[13] P. B. Andrews, *An Introduction to Mathematical Logic and Type Theory : To Truth through Proof* (Applied Logic Series ; 27), 2nd ed. Dordrecht: Kluwer Academic Publishers, 2002, ISBN: 1-4020-0763-9.

[14] F. D. Kamareddine, Twan. Laan ja R. P. Nederpelt, *A Modern Perspective on Type Theory : From Its Origins until Today* (Applied Logic Series ; v. 29). Dordrecht ; Kluwer Academic Publishers, 2004, ISBN: 978-1-4020-2334-7.

[15] G. Barthe ja T. Coquand, "An introduction to dependent type theory", teoksessa *Lecture Notes in Computer Science*, Berlin: Springer, 2002, s. 1–41, ISBN: 978-3-540-44044-4.

[16] Bart. Jacobs, *Categorical Logic and Type Theory* (Studies in Logic and the Foundations of Mathematics ; v. 141), 1st ed. Amsterdam ; Elsevier Science, 1999, ISBN: 0-08-052870-8.

[17] J. A. Robinson ja A. Voronkov, toim., *Handbook of Automated Reasoning*. Amsterdam: Elsevier, 2001, 1 s., ISBN: 978-0-262-18221-8 978-0-08-053279-0.

[18] R. Peikert, "Automated theorem proving: The resolution method", *ACM SIGSAM Bulletin*, vol. 21, nro 3, s. 61–68, elokuu 1987, ISSN: 0163-5824. DOI: 10.1145/29309.29319.

[19] Y.-R. Chen, S.-H. Chen ja S.-W. Lin, "SMT Solver With Hardware Acceleration", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, nro 6, s. 2055–2068, kesäkuu 2023, ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2022.3209550.

[20] A. Biere, toim., *Handbook of Satisfiability* (Frontiers in Artificial Intelligence and Applications 0922-6389 v. 185). Amsterdam Washington, DC: IOS Press, 2011, 966 s., ISBN: 978-1-58603-929-5 978-1-4416-1678-4 978-1-60750-376-7.

[21] J. Slagle ja S. Shankar, "Theorem Proving", teoksessa *Encyclopedia of Computer Science*, GBR: John Wiley and Sons Ltd., 2003, s. 1773–1776, ISBN: 0-470-86412-5.

[22] T. Kœhler, A. Goens, S. Bhat, T. Grosser, P. Trinder ja M. Steuwer, "Guided Equality Saturation", *Proceedings of the ACM on Programming Languages*, vol. 8, s. 1727–1758, POPL 2. tammikuuta 2024, ISSN: 2475-1421. DOI: 10.1145/3632900.

[23] L. De Moura ja N. Bjørner, "Engineering DPLL(T) + Saturation", teoksessa *Automated Reasoning*, A. Armando, P. Baumgartner ja G. Dowek, toim., vol. 5195, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 475–490, ISBN: 978-3-540-71069-1 978-3-540-71070-7. DOI: 10.1007/978-3-540-71070-7_40.

[24] I. Abdelaziz et al., "Learning to Guide a Saturation-Based Theorem Prover", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45,

nro 1, s. 738–751, 1. tammikuuta 2023, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2022.3140382.

[25] M. Echenim ja N. Peltier, "Combining Induction and Saturation-Based Theorem Proving", *Journal of Automated Reasoning*, vol. 64, nro 2, s. 253–294, helmikuu 2020, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-019-09519-x.

[26] L. Kovács ja A. Voronkov, "First-Order Theorem Proving and Vampire", teoksessa *Computer Aided Verification*, N. Sharygina ja H. Veith, toim., toim. D. Hutchison et al., vol. 8044, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 1–35, ISBN: 978-3-642-39798-1 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_1.

[27] J. Piepenbrock, T. Heskes, M. Janota ja J. Urban, "Guiding an Automated Theorem Prover with Neural Rewriting", teoksessa *Automated Reasoning*, J. Blanchette, L. Kovács ja D. Pattinson, toim., vol. 13385, Cham: Springer International Publishing, 2022, s. 597–617, ISBN: 978-3-031-10768-9 978-3-031-10769-6. DOI: 10.1007/978-3-031-10769-6_35.

[28] M. M. Kristiansen, *Proving theorems using deep learning*, NTNU, 2021.

[29] M. Hajdu, P. Hozzová, L. Kovács ja A. Voronkov. "Induction with Recursive Definitions in Superposition", esijulkaistu.

[30] W. Sieg ja F. Derakhshan, "Human-Centered Automated Proof Search", *Journal of Automated Reasoning*, vol. 65, nro 8, s. 1153–1190, joulukuu 2021, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-021-09594-z.

[31] L. J. Crotts ja S. Tate, "Comparison of Natural Deduction Theorem Provers used in Electronic Tutoring Systems", teoksessa *Proceedings of the 2022 6th International Conference on Education and E-Learning*, Yamanashi Japan:

ACM, 21. marraskuuta 2022, s. 43–49, ISBN: 978-1-4503-9842-8. DOI: 10.1145/3578837.3578844.

[32] G. Dowek. "Automated Theorem Proving in First-Order Logic modulo: On the Difference between Type Theory and Set Theory". versio 1, esijulkaistu.

[33] C. Rothgang, "Automated Theorem Proving for Dependent Typed Theories via a Translation to Higher-Order Logic", iii, 53, LII Seiten, 2023. DOI: 10.17169/REFUBIUM-44093.

[34] C. Rothgang, F. Rabe ja C. Benzmüller, "Dependently-Typed Higher-Order Logic", *ACM Transactions on Computational Logic*, s. 3 771 725, 14. lokakuuta 2025, ISSN: 1529-3785, 1557-945X. DOI: 10.1145/3771725.

[35] Z. Li et al. "A Survey on Deep Learning for Theorem Proving". versio 3, esijulkaistu.

[36] J. P. Bridge, S. B. Holden ja L. C. Paulson, "Machine Learning for First-Order Theorem Proving: Learning to Select a Good Heuristic", *Journal of Automated Reasoning*, vol. 53, nro 2, s. 141–172, elokuu 2014, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-014-9301-5.

[37] O. Kunčar ja A. Popescu, "From Types to Sets by Local Type Definition in Higher-Order Logic", *Journal of Automated Reasoning*, vol. 62, nro 2, s. 237–260, helmikuu 2019, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-018-9464-6.

[38] J. Meng ja L. C. Paulson, "Translating Higher-Order Clauses to First-Order Clauses", *Journal of Automated Reasoning*, vol. 40, nro 1, s. 35–60, tammikuu 2008, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-007-9085-y.

[39] T. Gauthier, C. Kaliszyk, J. Urban, R. Kumar ja M. Norrish, "TacticToe: Learning to Prove with Tactics", *Journal of Automated Reasoning*, vol. 65,

nro 2, s. 257–286, helmikuu 2021, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-020-09580-x.

[40] J. C. Blanchette, C. Kaliszyk, L. C. Paulson ja J. Urban, "Hammering towards QED", *Journal of Formalized Reasoning*, vol. Vol 9, 101–148 Pages, 29. tammikuuta 2016. DOI: 10.6092/ISSN.1972-5787/4593.

[41] Y. Jiang, *Machine learning for inductive theorem proving*, The University of Edinburgh, 2019.

[42] F. Wiedijk, "A Synthesis of the Procedural and Declarative Styles of Interactive Theorem Proving", *Logical Methods in Computer Science*, vol. Volume 8, Issue 1, s. 1046, 28. maaliskuuta 2012, ISSN: 1860-5974. DOI: 10.2168/LMCS-8(1:30)2012.

[43] J. Avigad, "Automated Reasoning for Mathematics", teoksessa *Automated Reasoning*, C. Benzmüller, M. J. Heule ja R. A. Schmidt, toim., vol. 14739, Cham: Springer Nature Switzerland, 2024, s. 3–20, ISBN: 978-3-031-63497-0 978-3-031-63498-7. DOI: 10.1007/978-3-031-63498-7_1.

[44] M. Pantsar, "Theorem proving in artificial neural networks: New frontiers in mathematical AI", *European Journal for Philosophy of Science*, vol. 14, nro 1, s. 4, maaliskuu 2024, ISSN: 1879-4912, 1879-4920. DOI: 10.1007/s13194-024-00569-6.

[45] A. Bundy, "Automated theorem provers: A practical tool for the working mathematician?", *Annals of Mathematics and Artificial Intelligence*, vol. 61, nro 1, s. 3–14, tammikuu 2011, ISSN: 1012-2443, 1573-7470. DOI: 10.1007/s10472-011-9248-8.

[46] M. H. Ter Beek et al., "Formal Methods in Industry", *Formal Aspects of Computing*, vol. 37, nro 1, s. 1–38, 31. maaliskuuta 2025, ISSN: 0934-5043, 1433-299X. DOI: 10.1145/3689374.

[47] G. O'Regan, *Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications* (Undergraduate Topics in Computer Science). Cham: Springer, 2017, 312 s., ISBN: 978-3-319-64020-4 978-3-319-64021-1. DOI: 10.1007/978-3-319-64021-1.

[48] S. Stock, J. Dunkelau ja A. Mashkoor, "Application of AI to formal methods — an analysis of current trends", *Empirical Software Engineering*, vol. 30, nro 6, s. 175, joulukuu 2025, ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-025-10729-8.

[49] M. Abdulaziz, "Interactive Theorem Provers: Applications in AI, Opportunities, and Challenges", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, nro 20, s. 22 660–22 660, 24. maaliskuuta 2024, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v38i20.30276.

[50] G. Mo, Y. Xiong, W. Huang ja L. Ma, "Automated Theorem Proving via Interacting with Proof Assistants by Dynamic Strategies", teoksessa *2020 6th International Conference on Big Data Computing and Communications (BIGCOM)*, Deqing, China: IEEE, heinäkuu 2020, s. 71–75, ISBN: 978-1-7281-8275-9. DOI: 10.1109/BigCom51056.2020.00016.

[51] M. L. Daggitt, W. Kokke, R. Atkey, L. Arnaboldi ja E. Komendantskya. "Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers". versio 1, esijulkaistu.

[52] A. De Almeida Borges et al., "Lessons for Interactive Theorem Proving Researchers from a Survey of Coq Users", *Journal of Automated Reasoning*, vol. 69, nro 1, s. 8, maaliskuu 2025, ISSN: 0168-7433, 1573-0670. DOI: 10.1007/s10817-025-09720-1.