Maxim Vitikainen

Induktiiviset tyypit

Tieto- ja ohjelmistotekniikan kandidaatintutkielma

24. kesäkuuta 2024

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Maxim Vitikainen

Yhteystiedot: maxvitik@student.jyu.fi

Ohjaajat: Sampsa Kiiskinen ja Antti-Jussi Lakanen

Työn nimi: Induktiiviset tyypit

Title in English: Inductive types

Työ: Kandidaatintutkielma

Sivumäärä: 21+0

Tiivistelmä: Induktiiviset tyypit ovat tapa mallintaa erilaisia tietotyyppejä kuten listoja, luonnollisia lukuja ja binäärihakupuita tyyppiteoriassa. Induktiivisia tyyppejä käytetään laajasti todistusassistenteissa erilaisten teoreemojen ja tietokoneohjelmien toteutukseen. Yleistys induktiivisista tyypeistä ovat korkeammat induktiiviset tyypit, jotka laajentavat induktiivisia tyyppejä mahdollistamalla tyypin alkioiden välisten ekvivalenssirelaatioiden määrittämisen. Tässä kirjallisuuskatsauksessa käsitellään induktiivisten ja korkeampien induktiivisten tyyppien teoriaa, niiden toteutuksia eri ohjelmointikielissä sekä erilaisia sovelluskohteita.

Avainsanat: induktiiviset tyypit, todistusassistentit

Abstract: Inductive types are a way to represent complex datatypes such as lists, natural numbers and binary search trees in type theory. Inductive types are currently widely used in proof assistants to implement mathemathical theorems and computer programs. A generalization of inductive types are higher inductive types, which extend inductive types by making it possible to define equivalence relations between inhabitants of a type. This thesis explores theory behind inductive and higher inductive types, their implementations in different programming languages and their use cases.

Keywords: inductive types, proof assistants

Sisällys

1 Johdanto

Induktiiviset tyypit (engl. *inductive types*) ovat algebrallisten tietotyyppien (engl. *algebraic datatypes*) yleistys, eli niillä voi mallintaa kaikkia tavallisten ohjelmointikielten tietotyyppejä. Ne helpottavat ohjelman oikeaksi todistamista, ja matematiikan kannalta tyyppiteoria, jossa on induktiiviset tyypit, on yksinkertainen perusta, jolla voi mallintaa hyvin suuren osan matematiikasta (Avigad 2023).

Induktiivinen tyyppi viritetään konstruktoreilla, jotka määrittävät tyypin rakenteen. Tyypiin konstruktoreista johtuvat eliminaattorit, jotka määrittävät sen, miten tyyppi vuorovaikuttaa muiden tyyppien kanssa ja millaisia operaatioita sillä voi tehdä (The Univalent Foundations Program 2013).

On olemassa myös korkeampia induktiivisia tyyppejä (engl. *higher inductive types*), joilla voi luoda tyypin alkioiden välisiä ekvivalenssirelaatioita. Korkeammat induktiiviset tyypit helpottavat joidenkin tyyppien määrittämisen ja esimerkiksi mahdollistavat todistamisen ekvivalenssiluokille (Fiore, Pitts ja Steenkamp 2022).

Curry–Howard-vastaavuus (engl. *Curry–Howard correspondence*) on matemaattinen teoria, jonka mukaan tyypit ovat väitteitä ja ohjelmat ovat todistuksia niille. Se mahdollistaa induktiivisten tyyppien käytön matemaattisissa todistuksissa. Tällä tavalla kirjoittamalla ohjelmia voidaan todistaa erilaisia matemaattisia teoreemoja ja todistaa oikeiksi erilaisia algoritmeja. Samalla tavalla voidaan esimerkiksi kirjoittaa ohjelma ja todistaa, että se toteuttaa tietyt vaatimukset, mikä on suotavaa kriittisissä järjestelmissä, joissa vaaditaan ohjelmien tarkkuutta ja vakautta (Leroy 2009; Chlipala 2013).

Induktiivisia tyyppejä käytetään paljon todistusassistenteissa, esimerkiksi Coq- ja Lean-todistusassistenttien standardikirjastojen monet tietorakenteet, kuten totuusarvot, luonnolliset luvut, listat ja rationaaliluvut, on toteutettu induktiivisina tyyppeinä (Avigad 2023; Chlipala 2013).

Tässä kirjallisuuskatsauksessa perehdytään induktiivisten ja korkeampien induktiivisten tyyppien teoriaan, niiden toteutuksiin ohjelmointikielissä Coq, Lean ja Agda, sekä

tutustutaan induktiivisten tyyppien reaalimaailman sovelluskohteisiin. Luvussa 2 käsitellään induktiivisten tyyppien teoria. Luvussa 3 esitellään erilaiset ohjelmointikielet, joissa on käytössä induktiiviset tyypit, sekä kirjastot, jotka laajentavat induktiivisia tyyppejä. Luvussa 3 esitellään induktiivisten tyyppien käyttökohteet. Lopuksi luvussa 4 kootaan tutkielman keskeiset asiat.

2 Teoria

Induktiivinen tyyppi on tyyppi $\mathbb{X}$, jonka voi mielivaltaisesti luoda rajallisesta joukosta konstruktoreita. Konstruktorit ovat kuvauksia, joiden maalijoukko on $\mathbb{X}$, ja joilla ei ole laskusääntöjä. Konstruktoreilla ei välttämättä ole parametreja, jolloin ne ovat vain $\mathbb{X}$:n alkioita (The Univalent Foundations Program 2013).

Induktiiviset tyypit ovat uniikkeja, eli yhden tyypin alkiot eivät voi olla toisen tyypin alkioita. Univalenteissa tyyppiteorioissa, kuten homotopiatyyppiteoriassa (engl. *homotopy type theory*), on kuitenkin univalenssiaksiooma

$$(A \simeq B) \to (A = B),$$

jonka avulla on mahdollista todeta kahden rakenteeltaan samanlaisten tyyppien olevan ekvivalentteja (The Univalent Foundations Program 2013).

2.1 Induktiiviset tyypit

Yksinkertaisin ei-triviaali induktiivinen tyyppi on luonnolliset luvut. Esimerkiksi seuraavat konstruktorit virittävät tyypin $\mathbb{N}$.

$$O : \mathbb{N}$$
$$S : \mathbb{N} \to \mathbb{N}$$

Tässä luonnolliset luvut muodostuvat alkupisteestä eli nollasta ja seuraavasta luvusta, jolloin:

$$S(O) \equiv 1,\ S(S(O)) \equiv 2,\ S(S(S(O))) \equiv 3...$$

Tässä onkin tyyppiteorian merkittävä ero joukko-opista. Joukko-opissa todistukset ovat usein muotoa: ”Onko olemassa alkio x, joka kuuluu joukkoon $\mathbb{X}$?” ja joukolle $\mathbb{X}$ on

annettu predikaatti, joka kertoo, kuuluuko alkio x joukkoon $\mathbb{X}$ sekä joukon $\mathbb{X}$ alkion rakenne. Tyyppiteoriassa sen sijaan ei ole olemassa predikaattia, joka kertoo kuuluuko alkio tyyppiin vai ei. On olemassa vain konstruktorit ja eliminaattorit, joilla tyypin alkiot muodostetaan ja muunnetaan. Konstruktorit ja eliminaattorit antavat päättely- ja laskusäännöt, joilla voidaan tehdä todistuksia.

Esimerkiksi on määritelty rationaalilukujen tyyppi $\mathbb{Q}'$. Olettaen, että on olemassa kokonaislukujen tyyppi $\mathbb{Z}$ ja positiivisten lukujen tyyppi $\mathbb{P}$, voidaan määritellä rationaalilukujen tyyppi $\mathbb{Q}'$ seuraavasti.

$$\frac{\square}{\square} : \mathbb{Z} \to \mathbb{P} \to \mathbb{Q}'$$

Tämän konstruktorin avulla on mahdollista todistaa, että jos $n : \mathbb{Z}$ ja $m : \mathbb{P}$, niin on olemassa rationaaliluku $\frac{n}{m} : \mathbb{Q}'$. Tämä määritelmä ainoastaan kertoo, että on olemassa alkiot muotoa $\frac{n}{m}$, eikä kerro mitään niiden suhteesta toisiin rationaalilukuihin. Tämä muodostuu ongelmaksi rationaaliluvuissa, koska $\frac{2}{4} \neq_{\mathbb{Q}'} \frac{1}{2}$, vaikka todellisuudessa ne ovat ekvivalentteja. Ongelma ratkeaa sillä, että luodaan uusi tyyppi $\mathbb{Q}$. Sen tehtävänä on muuttaa kaikki tyypin $\mathbb{Q}'$ alkiot normaalimuotoon jakamalla osoittajan ja nimittäjän niiden suurimmalla yhteisellä tekijällä. Tämä ratkaisu ei kuitenkaan ole kovin käytännöllinen, koska silloin lukuja pitää sieventää normaalimuotoon jokaisen laskutoimituksen jälkeen.

2.2 Korkeammat induktiiviset tyypit

Korkeammat induktiiviset tyypit ovat homotopiatyyppiteorian ja kuutiomaisen tyyppiteorian (engl. *cubical type theory*) merkittävä lisäys tyyppiteoriaan perustuksiin. Korkeammilla induktiivisilla tyypeillä on mahdollista mallintaa homotopiateorian rakenteita, tekijätyyppien (engl. *quotient types*) avulla virittää tyypin alkioden välisiä ekvivalenssirelaatioita, sekä toteuttaa muita monimutkaisempia tietorakenteita. Se tehdään lisäämällä tyypille konstruktorit, jotka määrittävät, millä tavalla tyypin alkiot ovat suhteessa toisiin alkioihin. (The Univalent Foundations Program 2013; Cohen ym. 2016;

Lumsdaine ja Shulman 2020)

Yhtenä hyvänä esimerkkinä korkeammat induktiiviset tyypit helpottavat rationaalilukujen määrittämistä. Sieventämisen sijaan rationaaliluvuille voidaan määritellä seuraava ekvivalenssirelaatio.

$$\sim : \mathbb{Q}' \to \mathbb{Q}' \to \mathscr{U}$$
$$\frac{n}{m} \sim \frac{k}{l} \equiv n \times l = m \times k$$

Tämän relaation avulla voidaan muodostaa tekijätyyppi $\mathbb{Q}/\sim$, joka edustaa rationaalilukujen tyyppiä.

Ekvivalenssirelaatio voidaan myös määrittää heti tyypin esittelyn yhteydessä.

$$\frac{\square}{\square} : \mathbb{Z} \to \mathbb{P} \to \mathbb{Q}$$
$$\mathrm{eqv}(q, q') : q \sim q' \to q = q'$$

3 Induktiiviset tyypit eri ohjelmointikielissä

Koska induktiivisia tyyppejä käytetään pääsääntöisesti todistusassistenteissa, ohjelmointikielissä induktiivisille tyypeille on asetettu tietyt rajoitteet. Yksi tällainen rajoite on tiukka positiivisuus (engl. *strict positivity*). Tyypin positiivisuus tarkoittaa sitä, että tyypin konstruktorin parametreissa määritettävä tyyppi ei saa esiintyä negatiivisilla paikoilla. Jos parametri on positiivisessa paikassa, negatiivinen paikka on nuolen vasemmalla puolella. Jos parametri on negatiivisessa paikassa, negatiivinen paikka on nuolen oikealla puolella. Tiukka positiivisuus tarkoittaa sitä, että tyyppi ei saa olleenkaan esiintyä parametreissa nuolen vasemmalla puolella (Dolan ym. 2024).

Tiukkaa positiivisuutta vaaditaan, koska jos funktion argumenteissa on määritettävä tyyppi, voi syntyä päättymätön rekursio jolloin mitä tahansa väitteitä voidaan todistaa. Tätä ilmiötä kutsutaan Curryn paradoksiksi (Chlipala 2013). Tämän takia kaikkien ohjelmien tulee todistusassistenteissa aina päättyä. Positiivisuuden vaatimuksen lisäksi on olemassa muita raijoitteita. Esimerkiksi jokaisen funktion pitää olla määritelty parametrien kaikilla arvoilla.

Chlipala (2013) antaa hyvän esimerkin induktiivisesta tyypistä, joka ei ole tiukasti positiivinen, ja eliminaattorista, joka aiheuttaisi päättymättömän rekursion.

```
Fail Inductive term : Set :=
| App : term -> term -> term
| Abs : (term -> term) -> term.

Fail Definition uhoh (t : term) : term :=
match t with
| Abs f => f t
| _ => t end.
```

Funktio `uhoh` kutsuu itseään loputtomasti, jos se on `Abs`-konstruktorin parametri, mikä johtaisi Curryn paradoksiin, jolloin voitaisiin todistaa mitä tahansa väitteitä.

Myös seuraava tyyppi ei ole sallittu, koska se ei ole tiukasti positiivinen. Vaikka `term` esiintyy ainoastaan positiivisilla paikoilla suhteessa koko lausekkeeseen, se on nuolen vasemmalla puolella, eli se on negatiivisella paikalla suhteessa omaan lausekkeeseen (Dolan ym. 2024).

```
Fail Inductive term : Set :=
| App : term -> term -> term
| Abs : ((term -> nat) -> term) -> term.
```

3.1 Coq

Coq on vuodesta 1984 kehitetty todistusassistentti, joka perustuu induktiivisten rakenteiden kalkyyliin (engl. *calculus of inductive constructions*). Induktiivisten rakenteiden kalkyylin kaksi perustaa ovat funktiotyypit ja induktiiviset tyypit (Chlipala 2013). Coq-kielellä luonnolliset luvut voidaan määritellä seuraavalla tavalla:

```
Inductive nat : Set :=
| O : nat
| S : nat -> nat.
```

Coq-kielessä on myös käytössä induktiivisista tyypeistä muutama erikoistapaus. Yksi erikoistapauksista ovat koinduktiiviset tyypit (engl. *coinductive types*), joilla on mahdollista määritellä äärettömiä tietorakenteita kuten esimerkiksi virtoja, joiden avulla voidaan toteuttaa laiska laskenta (engl. *lazy evaluation*). Chlipala (2013) antaa seuraavan esimerkin koinduktiivisesta tyypistä ja eräästä korekursiivisesta funktiosta:

```
CoInductive stream (A : Type) : Type :=
| Cons : A -> stream -> stream.
End stream.

CoFixpoint ones_zeroes : stream nat :=
  Cons _ 1 zeroes_ones
with zeroes_ones : stream nat :=
```

```
Cons _ 0 ones_zeroes.
```

`Cons`-konstruktoria hyödyntäen voidaan toteuttaa korekursiivinen funktio, joka palauttaa loputtomasti lukuja nolla ja yksi toistensa perään.

Äärettömistä virroista rekursiivisella funktioista voidaan ottaa äärellinen likiarvo. Chlipala (2013) antaa seuraavan esimerkin funktiosta, jolla voidaan ottaa äärettömästä virrasta n alkiota ja palauttaa ne listana:

```
Fixpoint approx A (s : stream A) (n : nat) : list A :=
  match n with
  | O => nil
  | S n' => match s with
            | Cons _ h t => h :: approx _ t n'
            end
  end.
```

Induktiivisten tyyppien toinen erikoistapaus ovat induktiiviset perheet (engl. *inductive families*), eli induktiiviset tyypit, jotka määrittävät muita tyyppejä toisen induktiivisen tyypin alkioiden avulla, eli induktiivinen perhe on tyyppiperhe, joka on indeksoitu induktiivisen tyypin alkioilla, ja se on yleistys parametrisoidusta tyypistä (Bauer 2018). Induktiivisilla perheillä voidaan mallintaa esimerkiksi erilaisia samankaltaisia tietorakenteita, jotka vaihtelee indeksin mukaan. Chlipala (2013) antaa esimerkin induktiivisesta perheestä, joka määrittelee pituus-indeksoidut listat. Tämä tyyppi on indeksoitu luonnollisen luvun mukaan, joka määrittele listan pituuden (Chlipala 2013).

```
Inductive ilist (A : Set) : nat -> Set :=
| Nil : ilist O
| Cons : forall n, A -> ilist n -> ilist (S n).
```

Coq-kielelle on myös olemassa kirjasto, joka toteuttaa ja formaalisti todistaa suurimman osan homotopiatyyppiteoriasta. Eli se toteuttaa korkeammat induktiiviset tyypit ja univalenssiaksiooman (Bauer ym. 2016).

3.2 Lean

Lean on vuonna 2013 julkaistu todistusassistentti. Kuten Coq, myös Lean pohjautuu induktiivisten rakenteiden kalkyyliin ja Leanissa on käytössä induktiiviset tyypit (Moura ja Ullrich 2021). Ominaisuuksiltaan Lean on hyvin lähellä Coq-kieltä. Coq-kielen ominaisuuksien lisäksi Leanissa on *propositional extensionality*-aksiooma, joka on erikoistapaus homotopiatyyppiteorian univalenssiaksioomasta:

$$(P \leftrightarrow Q) \rightarrow (P = Q)$$

Tämä tarkoittaa sitä, että kaksi propositiota ovat ekvivalentteja, jos ne implikoivat toisiaan. Tämän aksiooman avulla Lean toteuttaa tekijätyypit ilman korkeita induktiivisia tyyppejä (Avigad 2023). Tekijätyyppien lisäys ilman korkeampia induktiivisia tyyppejä on joidenkin tyyppiteorian tutkijoiden mielestä ristiriitaista, koska se rikkoo tiettyjä induktiivisten rakenteiden kalkyylin sääntöjä ja vaikeuttaa joissakin tapauksissa tyyppitarkistusta.

3.3 Agda

Agda on vuodesta 1990 kehitetty funktio-ohjelmointikieli ja todistusassistentti. Agda toteuttaa Martin-Löf tyyppiteorian ja lisää siihen muita konsepteja, joten Agdassa on käytettävissä induktiiviset ja koinduktiiviset tyypit. Niiden lisäksi Agdassa on käytössä induktiiviset perheet ja induktiiviset-rekursiiviset tyypit (engl. *inductive-recursive types*), jotka mahdollistavat tyypin alkioden ja eliminaattoreiden samanaikaisen määrittämisen, mikä mahdollistaa esimerkiksi uusien indeksoitujen universumien määrittämisen. Induktiivisia-rekursiivisia tyyppejä ei ole käytettävissä Coq-todistusassistentissa (Bove, Dybjer ja Norell 2009).

Agdalle on olemassa kirjastoja, jotka laajentavat Martin-Löf tyyppiteorian. Yksi sellainen kirjasto formalisoi suuren osan homotopiatyyppiteoriasta ja toinen toteuttaa kuutiomaisen tyyppiteorian.

Cubical on kuutiomainsen tyyppiteoriaan perustuva kirjasto, joka mahdollistaa kor-

keampien induktiivisten tyyppien käytön Agda-kielessä. Vezzosi (2021) antaa seuraavan esimerkin kokonaislukujen tyypistä, joka on määritelty Cubical-kirjaston korkeampien induktiivisten tyyppien avulla (Vezzosi, Mörtberg ja Abel 2021).

```
data Z : Set where
  pos : (n : N) -> Z
  neg : (n : N) -> Z
  posneg : pos 0 ≡ neg 0
```

Kuutiomaisen tyyppiteorian antamat korkeammat induktiiviset tyypit mahdollistavat sellaiset asiat, joiden toteuttaminen olisi muuten hankalaa tai jopa mahdotonta Agda-kielellä. Yksi asia, jota korkeammat induktiiviset tyypit mahdollistavat on homotopiateoriassa ja topologiassa käytettävien rakenteiden mallintaminen ja niiden ominaisuuksien todistaminen. Seuraavassa esimerkissa on määritelty ympyrä, joka on yksinkertaisin esimerkki tällaisesta homtopiateorian rakenteesta.

```
data S1 : Set where
  base : S1
  loop : base ≡ base
```

`base` on siis jokin ympyrän piste. Se on samanlainen konstruktori kuin esimerkiksi nolla luonnollisten lukujen tapauksessa. Tämä konstruktori luo uusia tyypin alkioita, ja sellaisia konstruktoreita kutsutaan pistekonstruktoreiksi. `loop` on konstruktori joka luo polun pisteestä takaisin pisteeseen luoden täten ympyrän. Konstruktoreita, jotka luovat polun alkiosta toiseen kutsutaan polkukonstruktoreiksi (The Univalent Foundations Program 2013).

4 Käyttökohteet ja sovellukset

Induktiivisten tyyppien käyttö ohjelmointikielissä rajoittuu todistusassistenteihin ja tutkimusohjelmointikieliin kuten edellä mainitut Agda, Coq ja Lean. Lisäksi ohjelmointikielet ja todistusassistentit kuten Idris (Brady 2013) ja Isabelle/HOL (Berghofer ym. 2023) käyttävät induktiivisia tyyppejä. Induktiivisia tyyppejä käytetään todistusassistenteissa, koska ne mahdollistavat erilaisten tietorakenteiden mallintamisen sellaisella tavalla, jolla tyyppitarkistin osaa tarkistaa tyyppien, funktioiden ja todistuksien oikeellisuuden.

Koska suuri osa todistusassistenteista käyttää induktiivisia tyyppejä, lähes kaikessa formaalissa tietokoneavusteisessa todistamisessa käytetään induktiivisia tyyppejä. Tämän takia vaikka induktiivisten tyyppien käyttö on hyvin rajallista, niiden vaikutus ulottuu pitkälle. Induktiivisten tyyppien avulla voi mallintaa kaikki yleisessä käytössä olevien ohjelmointikielten tietotyypit, jolloin induktiivisten tyyppien avulla on mahdollista kehittää ja mallintaa ohjelmia, joita käytetään oikean maailman tehtävissä.

4.1 Matematiikka

Todistusassistenteilla on todistettu useita matemaattisia teoreemoja, joiden todistaminen ilman tietokonetta olisi erittäin työlästä tai jopa mahdotonta, kuten esimerkiksi neljän värin teoreema (Gonthier 2008). Ennen formaalia todistusta teoreema on yritetty todistaa tapausanalyysillä, jossa tietokoneen avulla on käyty läpi yli 10 000 tapausta. Tällaisten todistuksien tekeminen ja vertaisarviointi on hyvin työläs prosessi ja täytyy luottaa siihen, että ohjelma on kirjoitettu oikein. Gonthierin (2005) todistuksessa Coq-todistusassistentin tyyppitarkistin varmistaa sen, että kirjoitettu ohjelma on oikein, mikä lisää todistuksen uskottavuutta ja luotettavuutta (Gonthier 2005).

Yksi toinen viime aikojen merkittävä todistusprojekti on Lean-yhteisön aloittama projekti, jonka tarkoitus on formalisoida mahdollisimman paljon matematiikka Lean-todistusassistentilla. Projektiin osallistuu suuri määrä ihmisiä, joilla on eri taustat ja osaamistasot, mikä perinteisessä tapauksessa tekisi todistuksien tarkistuksesta hyvin

työlään. Todistusassistentti kuitenkin takaa sen, että todistus on tehty oikein, eli ei tarvitse luottaa siihen, että ihminen on tehnyt todistuksen oikein, sen sijaan täytyy luottaa ainoastaan siihen, että todistusassistentti on oikeassa, mikä mahdollistaa näin suuren mittakaavan yhteistyön eritaustaisten ihmisten välissä matemaattisessa todistusprojektissa (Doorn, Ebner ja Lewis 2020).

Koska korkeammat induktiiviset tyypit ovat tulos homotopian ja tyyppiteorian yhdistämisestä, Korkeammilla induktiivisilla tyypeillä on myös luontevaa esittää homotopian rakenteita (The Univalent Foundations Program 2013). Esimerkiksi Cubical Agdan korkeampien induktiivisten tyyppien avulla on formaalisti tarkistettu Guillaume Brunieren väitöskirjan seuraava väite: $\pi_4(S^3) \simeq \mathbb{Z}/2\mathbb{Z}$ (Ljungström ja Mörtberg 2024).

4.2 Ohjelmointi

Tietotekniikan maailmassa Xavier Leroy vuonna 2009 kehitti formaalisti todistetun CompCert C-kielen kääntäjän, jonka todistamisessa on käytetty induktiivisia tyyppejä esimerkiksi C-kielen syntaksipuun mallintamiseen (Leroy 2009). Tämä kääntäjä takaa sen, että kääntäjä kääntää ohjelman oikein. Kääntäjä ei kuitenkaan varmista sitä, että ohjelma linkittyy oikein, tai sen, että ohjelmassa ei ole muita ongelmia. Ainoastaan käännösvaihe on todistettu.

Ohjelmien formaali todistaminen on hyvin työlästä ja se ei poista kaikkia ohjelmistokehityksen ongelmia. Vaikka ohjelma olisi formaalisti todistettu, täytyy silti luottaa esimerkiksi Coq-kielen ytimeen ja siihen, että Coq-kääntäjä tuottaa korrektia koodia. Coq-ytimessä on löydetty kriittisiä ongelmia noin kerran vuodessa (Monniaux ja Boulmé 2022). Chlipala (2013) kuitenkin väittää, että todistettu ohjelma kuitenkin on astetta parempi, kuin sellainen jonka toiminnasta ei ole formaalia todistusta (Chlipala 2013). Tästä väitteestä löytyy näytteitä käytännön sovelluksista. CompCert-kääntäjästä on löydetty paljon vähemmän ongelmia, kuin muista kääntäjistä (Monniaux ja Boulmé 2022).

CompCertin lisäksi on olemassa muita isoja ohjelmia, joita on formaalisti todistettu, esimerkiksi seL4 käyttöjärjestelmäydin, joka on todistettu Isabelle/HOL todistusassis-

tentilla. seL4 on mikrokäyttöjärjestelmäydin, joka on nykyään käytössä monissa kriittisissä tehtävissä. Projektin tavoite on kehittää käyttöjärjestelmäydin, jolla on hyvä suorituskyky ja tietoturva. seL4 on todistettu toimivaksi käytännössä. Esimerkiksi laitteet, joita on muokattu käyttämään seL4 käyttöjärjestelmäydintä, ovat olleet paljon kestävämpia kyberhyökkäyksiä vastaan (Andronick 2019).

Todistusassistentteja ja induktiivisia tyyppejä on myös sovellettu tekoälyn kanssa. OpenAI on tutkinut todistusassistentin yhdistämistä kielimalliin, jolloin kielimalli kykeni suoriutumaan matematiikan ja logiikan testeistä paremmin (Polu ym. 2022). Lisäksi Meta on tutkinut todistuksien automatisointia tekoälyn avulla käyttäen *HyperTree* todistuksenhakumenetelmää (Lample ym. 2022).

Tutkimusprojektien lisäksi induktiivisia tyyppejä käytetään kaupallisessa ohjelmistossa. Esimerkiksi Bedrock Systems käyttää Coq-todistusassistenttia omien ja asiakkaiden palveluiden tietoturvan parantamiseksi. Yksi toinen esimerkki on Formal Land, joka tarjoaa ohjelmien oikeaksitodistamisen palveluna. Heidän merkittävin projekti tähän asti on ollut Tezos-lohkoketjun protokollan muodollinen varmistaminen.

5 Yhteenveto

Induktiiviset typpit ilmaisevat erilaisia tietorakenteita ja niillä on päättely- ja laskusäännöt, joiden avulla on mahdollista todistaa tiettyjä väitteitä, minkä ansiosta induktiivisia tyyppejä käytetään paljon todistusassistenteissa kuten Coq, Agda ja Lean. Homotopiatyyppiteoria ja myöhemmin kehitetty kuutiomainen tyyppiteoria laajentavat induktiiviset tyypit korkeammilla induktiivisilla tyypeillä, joiden avulla on mahdollista tehdä tyypin alkioiden välisiä ekvivalenssirelaatioita, mikä mahdollistaa homotopian rakenteiden mallintamisen, helpomman tyyppien määrittelyn tietyille tyypeille, ekvivalenssiluokille todistamisen sekä helpomman todistuksien korjaamisen.

Matematiikassa on todistettu erilaisia teoreemoja niiden avulla ja varsinkin homotopiateoriassa niille on löytynyt paljon käyttöä. Tietojenkäsittelytieteessä induktiivisia tyyppejä on käytetty esimerkiksi CompCert kääntäjässä C-ohjelmointikielen erilaisten rakenteiden mallintamiseen. Tämä helpottaa ohjelman oikeaksi todistamista, mikä helpottaa isojen todistettujen järjestelmien kehittämisen, mikä vuorostaan vähentää merkittävästi ohjelmassa olevien virheiden määrän, sillä tyyppitarkistin löytää suuren osan virheistä ja tarkistaa tehdyt todistukset.

Tulevaisuudessa kun todistuksien automatisointi tai tekoälyavusteinen todistaminen kehittyy, yhä useampi kriittinen järjestelmä tulee olemaan muodollisesti mallinnettu tai varmistettu, koska suuri automaatioaste tai tekoälyn avustus tulevat helpottamaan todistamista merkittävästi. Induktiivisten tyyppien yleistymisen myötä funktio-ohjelmointikielet tulevat olemaan joustavempia ja varmempia, sillä induktiivisten tyyppien avulla ohjelmoijalla on enemmän vapautta määritellä, miten tyyppi käyttäytyy ja tyyppitarkistin pystyy paremmin tarkistaa ohjelman oikeellisuuden.

Lähteet

Andronick, June. 2019. “Successes in Deployed Verified Software (and Insights on Key Social Factors)” [kielellä en]. Teoksessa *Formal Methods – The Next 30 Years,* toimittanut Maurice H. Ter Beek, Annabelle McIver ja José N. Oliveira, 11800:11–17. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-030-30941-1 978-3-030-30942-8, viitattu 15. huhtikuuta 2024. https://doi.org/10.1007/978-3-030-30942-8_2.

Avigad, Jeremy. 2023. “Theorem Proving in Lean”, https://leanprover.github.io/theorem_proving_in_lean/.

Bauer, Andrej. 2018. *In Agda's GADT, are "parameterized"and "indexed"different semantically?* https://cs.stackexchange.com/q/98546.

Bauer, Andrej, Jason Gross, Peter LeFanu Lumsdaine, Mike Shulman, Matthieu Sozeau ja Bas Spitters. 2016. *The HoTT Library: A formalization of homotopy type theory in Coq* [kielellä en]. ArXiv:1610.04591 [cs, math], joulukuu. Viitattu 25. maaliskuuta 2024. http://arxiv.org/abs/1610.04591.

Berghofer, Stefan, Tobias Nipkow, Lawrence C. Paulson ja Markus Wenzel. 2023. “Examples of Inductive and Coinductive Definitions in HOL” [kielellä en], https://isabelle.in.tum.de/library/HOL/HOL-Induct/document.pdf.

Bove, Ana, Peter Dybjer ja Ulf Norell. 2009. “A Brief Overview of Agda – A Functional Language with Dependent Types” [kielellä en]. Teoksessa *Theorem Proving in Higher Order Logics,* toimittanut Stefan Berghofer, Tobias Nipkow, Christian Urban ja Makarius Wenzel, 5674:73–78. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-03358-2 978-3-642-03359-9, viitattu 2. helmikuuta 2024. https://doi.org/10.1007/978-3-642-03359-9_6.

Brady, Edwin. 2013. “Idris, a general-purpose dependently typed programming language: Design and implementation” [kielellä en]. *Journal of Functional Programming* 23, numero 5 (syyskuu): 552–593. ISSN: 0956-7968, 1469-7653, viitattu 31. maaliskuuta 2024. https://doi.org/10.1017/S095679681300018X. https://www.cambridge.org/core/product/identifier/S095679681300018X/type/journal_article.

Chlipala, Adam. 2013. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant* [kielellä en]. The MIT Press, joulukuu. ISBN: 978-0-262-31787-0, viitattu 6. helmikuuta 2024. https://doi.org/10.7551/mitpress/9153.001.0001.

Cohen, Cyril, Thierry Coquand, Simon Huber ja Anders Mörtberg. 2016. *Cubical Type Theory: a constructive interpretation of the univalence axiom* [kielellä en]. ArXiv:1611.02108 [cs, math], marraskuu. Viitattu 2. helmikuuta 2024. http://arxiv.org/abs/1611.02108.

Dolan, Stephen, Andrej Bauer, Leo White ja Jeremy Yallop. 2024. *Counterexamples in Type Systems.* https://counterexamples.org/.

Doorn, Floris van, Gabriel Ebner ja Robert Y. Lewis. 2020. “Maintaining a Library of Formal Mathematics” [kielellä en], 12236:251–267. ArXiv:2004.03673 [cs, math]. Viitattu 15. huhtikuuta 2024. https://doi.org/10.1007/978-3-030-53518-6_16.

Fiore, Marcelo P., Andrew M. Pitts ja S. C. Steenkamp. 2022. “Quotients, inductive types, and quotient inductive types” [kielellä en]. *Logical Methods in Computer Science* Volume 18, Issue 2 (kesäkuu): 7076. ISSN: 1860-5974, viitattu 2. helmikuuta 2024. https://doi.org/10.46298/lmcs-18(2:15)2022.

Gonthier, Georges. 2005. “A computer-checked proof of the Four Color Theorem” [kielellä en]. Hal-04034866, https://inria.hal.science/hal-04034866.

———. 2008. “Formal Proof—The Four- Color Theorem” [kielellä en]. *Notices of the ACM* 55 (11): 1382–1393.

Lample, Guillaume, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez ja Timothée Lacroix. 2022. *HyperTree Proof Search for Neural Theorem Proving* [kielellä en]. ArXiv:2205.11491 [cs], toukokuu. Viitattu 11. huhtikuuta 2024. http://arxiv.org/abs/2205.11491.

Leroy, Xavier. 2009. "Formal verification of a realistic compiler" [kielellä en]. *Communications of the ACM* 52, numero 7 (heinäkuu): 107–115. ISSN: 0001-0782, 1557-7317, viitattu 4. maaliskuuta 2024. https://doi.org/10.1145/1538788.1538814.

Ljungström, Axel ja Anders Mörtberg. 2024. *Formalising and Computing the Fourth Homotopy Group of the 3-Sphere in Cubical Agda* [kielellä en]. ArXiv:2302.00151 [cs, math], huhtikuu. Viitattu 18. kesäkuuta 2024. http://arxiv.org/abs/2302.00151.

Lumsdaine, Peter Lefanu ja Michael Shulman. 2020. "Semantics of higher inductive types" [kielellä en]. *Mathematical Proceedings of the Cambridge Philosophical Society* 169, numero 1 (heinäkuu): 159–208. ISSN: 0305-0041, 1469-8064, viitattu 2. helmikuuta 2024. https://doi.org/10.1017/S030500411900015X.

Monniaux, David ja Sylvain Boulmé. 2022. "The Trusted Computing Base of the CompCert Verified Compiler" [kielellä en]. Teoksessa *Programming Languages and Systems,* toimittanut Ilya Sergey, 13240:204–233. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-030-99335-1 978-3-030-99336-8, viitattu 15. huhtikuuta 2024. https://doi.org/10.1007/978-3-030-99336-8_8.

Moura, Leonardo de ja Sebastian Ullrich. 2021. "The Lean 4 Theorem Prover and Programming Language". Teoksessa *Automated Deduction – CADE 28,* toimittanut André Platzer ja Geoff Sutcliffe, 625–635. Cham: Springer International Publishing. ISBN: 978-3-030-79876-5.

Polu, Stanislas, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin ja Ilya Sutskever. 2022. *Formal Mathematics Statement Curriculum Learning* [kielellä en]. ArXiv:2202.01344 [cs], helmikuu. Viitattu 11. huhtikuuta 2024. http://arxiv.org/abs/2202.01344.

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics* [kielellä en]. Institute for Advanced Study. https://homotopytypetheory.org/book.

Vezzosi, Andrea, Anders Mörtberg ja Andreas Abel. 2021. "Cubical Agda: A dependently typed programming language with univalence and higher inductive types" [kielellä en]. *Journal of Functional Programming* 31:e8. ISSN: 0956-7968, 1469-7653, viitattu 2. helmikuuta 2024. https://doi.org/10.1017/S0956796821000034.