Suunnitteludokumentti

Ohjelmistotuotantoprojekti Todistustoimitin

Teppo Kankaanpää, Jaakko Nenonen, Aki Nyrhinen, Visa Röyskö, Juha-Matti Tapio

13. maaliskuuta 2003
Helsingin yliopisto
Tietojenkäsittelytieteen laitos

$Id: suunnitteludokumentti.tex,v 1.10 2003/03/13 04:21:48 jmtapio Exp $

Sisältö

Kuvat

Liitteet

1. Eka liite

Historia

Päiväys	Versio	Kuvaus	Tekijät
13.3.2003	0.1	Ensimmäinen luonnos	Kaikki

1 Yleistä

Viime syksynä luennoitiin laitoksella kurssi Proof Theory and Proof Search, jossa käsiteltiin luonnollisen logiikan tapaista formalismia, jonka etuina tähän nähden on se, että todistamisprosessi vaatii vähemmän intuitiivista päättelyä. Lauseen todistaminen tapahtuu ”käänteisessä järjestyksessä”, lähtien lopputuloksesta ja jakaen sitä yhä pienempiin osiin sääntöjä käyttäen kunnes päädytään aksioomiin tai erityisiin kohtiin, joissa ollaan päädytty ”identtisesti tosiin” lauseisiin, jolloin todistus on valmis.

Aiemmin kurssin käytössä on ollut vain tekstipohjaisia ohjelmia. Todistustoimittimen on graafinen helppokäyttöinen ohjelma, jolla todistuksia voidaan laatia mukaillen sitä, millä tavalla todistukset kirjoitettaisiin paperille.

1.1 Ohjelmiston rakenne

Todistustoimitin jakautuu pääpiirteissään neljään osajärjestelmään: käyttöliittymään, käsiteltävään todistukseen eli tietorakenteisiin, jäsentäjiin ja kalkyyliin, joka käyttöliittymän käskystä tekee muutoksia tietorakenteisiin. Sekä käyttöliittymä että kalkyyli pääsevät käsiksi suoraan tietorakenteeseen, sillä sama tietorakenne pitää sisällään myös ruutuesitykselle olennaisia tietoja, kuten esimerkiksi kunkin todistuksen osan ympäröivän neliön mitat.

Luonnollisesti ajateltuna todistukset ja kaavat jäsentäjineen kuuluisivat osina kalkyyliin ja käyttöliittymä käsittelisi niitä kalkyylin rajapinnan läpi. Tässä suunnitelmassa tietorakenteet eivät kuulu ainoastaan kalkyylille siksi, että muussa tapauksessa käyttöliittymä joutuisi järjestämään teennäisen kirjanpidon tiedoille, jotka sopivat tallennettaviksi samaan tietorakenteeseen sen datan kanssa, jota ne kuvaavat.

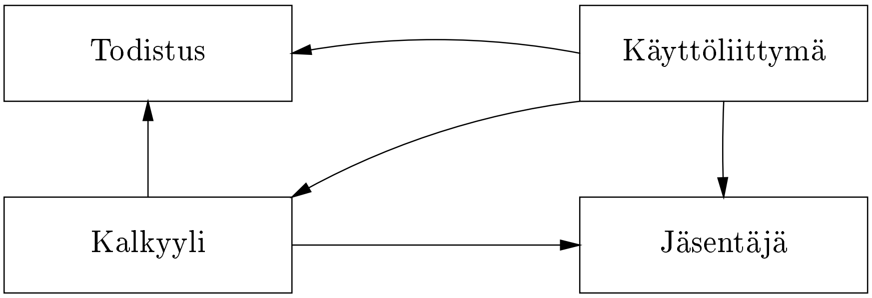
1.2 Suunnittelufilosofia

Suunnittelussa on pyritty välttämään tarpeetonta pikkutarkkutta luokkien ja toimintojen kuvaamisessa. Aksessoreita ei tulla ohjelmoimaan ohjelmoinnin itsensä vuoksi, vaan pyritään siihen, että pääosa kirjoitetusta koodista on hyödyllistä ja ymmärrettävää.

Myöskään olioajattelua ei yritetä venyttää äärimmilleen. Proseduraalista tai funktionaalista ajattelutapaa siis käytetään häpeilemättä silloin kun se tuntuu tuottavan yksinkertaisemman ja toimivamman ratkaisun kuin oliomalli.

2 Arkkitehtuuri

Todistustoimitin jakautuu toiminnallisesti neljään osajärjestelmään: kalkyyliin, jäsentäjään, käyttöliittymään ja todistukseen. Osajärjestelmät eivät ole erityisen itsenäisiä. Erityisen vahva sidonta on kalkyylin, todistuksen ja jäsentäjän välillä. Voisi sanoa että todistus ja jäsentäjä ovat kalkyylin osajärjestelmiä, mutta myös käyttöliittymällä on tehokkuussyistä pääsy suoraan todistukseen kulkematta kalkyylin kautta. Oheisesta kuvassa 1 olevasta osajärjestelmäkaaviosta nähdään, miten huonosti järjestelmä jakautuu itsenäisiin komponentteihin.



Kuva 1: Osajärjestelmäkaavio

2.1 Kalkyyli

Kalkyylin ensisijainen tehtävä on todistuspuun muokkaaminen siten että todistuspuu säilyy oikeellisena. Tarkennettuna kalkyyli tarjoaa käyttäjälleen (luokka) riittävän vahvan mutta tarpeeksi rajoittavan rajapinnan muokata todistusta niissä puitteissa joissa sen tulee olla kyseisellä kalkyylillä mahdollista. Teknisenä yksityiskohtana kalkyyli toteuttaa myös yksitasoisen undo-operaation, jolla voidaan perua edellinen tehty operaatio (myös undo voidaan perua).

Kalkyyli tarvitsee jäsentäjää luodakseen käsiteltävän todistuksen.

2.2 Todistus

Todistuksen tehtävä on esittää todistuksen tila kullakin hetkellä, ja tallentaa muiden järjestelmien tarvitsemaa tietoa (koordinaatteja, kaavojen muoto, jne). Erityispiirteenä todistus ei koskaan unohda todistuspuun haaroja joita siihen on kalkyylin avulla rakennettu. Tämä erityispiirre on tärkeä järjestelmän undo-vastineen kannalta.

2.3 Käyttöliittymä

Käyttöliittymän tehtävä on esittää todistus2.2 käyttäjälle ymmärrettävässä muodossa sekä tarjota sitä käyttäen käyttäjälle helppokäyttöinen ja ymmärrettävä käyttörajapinta kalkyylille. Käyttöliittymä valitsee jäsentäjän, jota kalkyyli käyttää todistuksen jäsentämiseen.

2.4 Jäsentäjä

Jäsentäjän tehtävä on hieman harhaanjohtavasti paitsi todistuksen2.2 tuottaminen sille määritellystä merkkijonoesityksestä3.1.1, myös tarjota kuvaus takaisin samaan esitykseen. Jäsentäjä vastaa siis eräässä mielessä tallennuksesta ja latauksesta, ja on siten hyvin tiiviisti todistukseen liittyvä.

3 Luokkarakenne

Liitteessä X on kuvattu ohjelman luokkarakenne UML-kaavioina. Tässä kappaleessa rakenne ja sen suunnittelun motiivit kuvataan tarkemmin.

Ohjelma jakautuu neljään enemmän tai vähemmän itsenäiseen komponenttiin: jäsentäjään, todistusrakenteen tallentajaan, kalkyylin toteuttajaan ja käyttöliittymään.

3.1 Jäsentäjä

Jäsentäjän tehtävänä on tuottaa annetusta merkkijonosta kokoelma jäsennyspuita. Sitä käytetään ohjelmassa kolmessa tapauksessa. 1) Kun käyttäjä luo uuden todistuksen ja kirjoittaa lauseen ohjelmaan, jäsentäjä kääntää sen ohjelman käyttämään muotoon. 2) Kun kaava ladataan levyltä, jäsentäjä jäsentää sen. Tallennustiedoston formaatti on merkkijonona, joka pitää kääntää ohjelman käyttämään muotoon. 3) Komponentti tuottaa annetusta kaavasta merkkijonoesityksen jolloin voidaan yhdessä komponentissa pitää huolta siitä, että lataus- ja tallennus on yhtenäinen.

3.1.1 Merkkijonon muoto

Jäsentäjälle annettava merkkijono noudattaa seuraavaa EBNF-muotoista[1] määritystä:

```
proof := prmultiset ',' prformula '{' SP* sequent SP* proof* '}'
sequent := '[' SP* multiset+ SP* ']'
multiset := '[' SP* formula* SP* ']'
formula := '(' SP* (formula SP+)? connective SP+ formula SP* ')' | atom
atom := [A-Z]([a-z][A-Z])*
connective := [a-z]+
prmultiset := [0-9]+
prformula := [0-9]+
```

Minkä tahansa kahden tokenin välillä voi olla mielivaltainen määrä välilyöntejä. Muita tyhjämerkkejä ei oteta huomioon.

Kuten ylläolevasta näkee, konnektiivit ja propositiot voivat olla mitä muotoa tahansa. Jäsentäjä ei ota jäsennysvaiheessa kantaa kaavan semantiikkaan muuten kuin erottelemalla propositiot ja konnektiivit toisistaan niin että propositiot alkavat aina isolla kirjaimella.

3.1.2 Luokat

Jäsentäjä koostuu kahdesta luokasta: *ParsedFormula* ja *Sequent*. ParsedFormula on oksa- ja lehtisolmu binääripuussa, joka on jäsennetyn kaavan esitys. Sequent sisältää kaksi monijoukkoa viittauksia ParsedFormuloihin, eli monijoukon kaavojen jäsennyspuiden juuriin.

[1] Extended Backus-Naur Form

Sequent-luokalla on vähän merkitystä jäsentäjän ulkopuolella. Sen sijaan ParsedFormula osallistuu myös tietosisällön esittämiseen. Julkiset rutiinit ovat Sequent-luokassa, mutta ParsedFormula sisältää myös koodia, joka tuottaa yhden monijoukosta erotetun kaavan esityksestä ParsedFormulan, tai ParsedFormulasta merkkijonon. Sequentissa on monijoukon käsittelyyn liittyvä osa.

3.2 Tietosisältö

Tietosisällöltään sovellus jakautuu kalkyyleihin (calculus-aliluokat), tietorakenteisiin (proof, formula, parsedformula) ja erilaisiin tietorakenteiden rakentajiin ja tallentajiin (proofparser, sequentparser aliluokat), jotka osaavat joko luoda tietorakenteen tietovirrasta tai tallentaa tietorakenteen tietovirraksi.

3.2.1 Calculus

Koska kalkyylejä on useita, on luontevaa määritellä kalkyylille rajapinta, jonka varsinaiset kalkyylin toteuttavat luokat perivät, ja esiintyvät varsinaisessa toimintatilanteessa rajapinnan nimellä.

Jokaisen kalkyylin tulee toteuttaa seuraavat toiminnot:

```
new(ProofParser pp, Stream in) throws ParseError, IOException
```

New:llä tarkoitetaan luokan konstruktoria. Jokaisen kalkyylin tulee toteuttaa konstruktori, joka käyttää parametrina saamaansa jäsentäjää todistuspuun tuottamiseen tietovirrasta. Käyttäjän aloittaessa uutta todistusta kyseessä tietovirta on käyttäjän syöttämästä kaavasta pienellä käsittelyllä muodostettu todistuspuuksi jäsentyvä stringstream. Jo tallennetun todistuksen avaamisen tapauksessa tietovirta on avattu tiedosto. Tietovirtaan sopivan jäsentäjän valitseminen jää käyttäjän vastuulle. Jos tietovirta ei ole jäsentyvä, jäsentäjä heittää poikkeuksen, joka on ParseError-poikkeuksen aliluokka. Tietovirtojen käsittelystä voi myös tulla poikkeustilanteita, joiden käsittelyä pyritään välttämään itse jäsentäjissä, ja palauttamaan poikkeuksia korkeammille tasoille ja lopulta kutsujalle.

Jos tarpeen, kalkyylien toteutukset voivat suorittaa ParsedFormuloille manipulaatiota ennen todistuspuun alkioiden luontia. Tyypillisesti tämä manipulointi lienee konnektiivinimien korvaamista vastaavilla yksikäsitteisillä olioviitteillä vertailuoperaatioiden tehostamiseksi. Jatkokehittelyn vuoksi mahdollisuus on tarpeen, koska leikkaussäännön (cutrule) toteuttaminen on mahdollista siten toteuttamalla uusien atomien esittelyn lisäksi atomin korvaaminen kaavalla, ja tällöin on miellyttävää, että kaavan atomi-instanssit ovat yksikäsitteisiä. Tämä yksikäsitteistämisvaihe on siis jätettävä mahdolliseksi kalkyylille.

Calculus on rajapintaluokka, jonka varsinaiset kalkyylit (G3IPCalculus, G3CPCalculus, ...) perivät.

```
apply(Proof proof, int formula, int multiset)
```

Apply soveltaa kalkyyliä valitun todistuksen (proof) valittuun kaavaan (formula, multiset). On selvää, että kutsujalla täytyy olla pääsy kalkyylin käyttämään todistuspuuhun, jotta tätä metodia voitaisiin ylipäätään käyttää.

Apply metodin kuvaus pseudojaaavalla seuraa

```
sotilas kohisee Formula[][] mkseq(ArrayList a,b):
    Formula r = new Formula[2][];

    r[RIGHT] = new Formula[a.size()];
    for (i = 0; i < a.size(); ++i)
        r[RIGHT][i] = (Formula)a.elementAt(i);

    r[LEFT] = new Formula[b.size()];
    for (i = 0; i < b.size(); ++i)
        r[LEFT][i] = (Formula)b.elementAt(i);

    return r;

kuplii ja kohisee tyhjää apply(p, f, m):
    int i;
    Formula[][] seq = p.children;
    Formula f = seq[m][f];
    ParsedFormula pf = f.parsetree;

    if (m == RIGHT)
        if (pf.data.instanceOf("String"))
            error("ÄLÄHÄN SOHAA MUN ATOMEJA");

        else if (pf.data == implication)
            ArrayList left = new ArrayList(seq[LEFT]);
            ArrayList right = new ArrayList(seq[RIGHT]);

            left.addLast(new Formula(pf.left));
            right.insert(f, new Formula(pf.right));

            p.addChildren(m, f, 0, mkseq(left, right));

        else if (pf.data == disjunction)
            ...
         ...
    else if (m == LEFT)
         ...

undo(void)
```

Undo peruuttaa edellisen tehdyn *apply*:n. Undo on toteutettu siten, että kalkyylissä on undoa varten kentät joiden sisältönä on viimeksi käsitelty proof-instanssi, ja sitä koskevat kokonaisluvut (taulukoiden indeksit) multiset ja formula, ja formulaa koskevasti vastaava rule (sovelletun säännön kalkyylikohtainen järjestysnumero).

3.2.2 Proof

Todistus (Proof) ja kaava (Formula) muodostavat yhdessä todistuspuun rakenteen. Syy siihen että rakenteen rekursio kiertää kaavan kautta on undo-operaation toteutus. Kalkyyleissä voi nimittäin olla useita sääntöjä joita voidaan soveltaa samaan kaavaan, jolloin sovelletun säännön numeron tallentaminen todistukseen olisi tarpeettoman monimutkaista.

Konseptuaalisesti rekursio tapahtuu indeksoimalla neliulotteisen taulukon kolmea ensimmäistä ulottuvuutta. Ensimmäisellä indeksillä valitaan monijoukko (multiset), joka on tyypilisesti (ainakin kaikissa kirjoittajan tuntemissa sekvenssikalkyyleissä) vain kaksiarvoinen (oikea, vasen). Toinen indeksi valitsee prinsipaalisen kaavan valitun sekvenssin sisältä (kalkyyli pystyy jatkamaan todistusta näiden indeksien varassa). Kolmannella indeksillä valitaan sovellettu sääntö siitä sääntöjen joukosta jota kyseiseen kaavaan kyseisessä monijoukossa voidaan soveltaa. Viimeinen ulottuvuus numeroi todistuspuun haarat, joita tunnetuissa kalkyyleissä on korkeintaan kaksi.

Proof sisältää seuraavanlaisia metodeja.

```
new(Formula sequent[][])
new(ParsedFormula sequent[][])
```

Peruskontruktori luo uuden todistuksen, joka sisältää parametrina annetun sekvenssin, eli 2-ulotteisen taulukon kaavoja. Variaatio luo itse Formula-objektit, mutta ottaa parametrina jäsennyspuut.

```
AddChildren(int multiset, int formula, int rule, Proof children[])
removeChildren(int multiset, int formula, int rule)
```

AddChildren lisää parametrina saadut alipuut (children) indeksien määrittämään paikkaan taulukossa. *RemoveChildren* on vastaava poisto-operaatio.

```
getVisibleChildren(void)
```

GetVisibleChildren palauttaa tällä hetkellä näkyvissä olevat todistuksen haarat, jotka ovat jatkoa kyseiselle solmulle.

```
int setVisibleFormula(int)
int setVisibleMultiset(int)
```

SetVisibleMultiset ja *GetVisibleMultiset* asettavat näkyvän monijoukon ja kaavanumeron kyseisen monijoukon sisällä. Paluuarvo on kyseisen attribuutin vanha arvo, jolla saman metodin kutsuminen palauttaa tilanteen sellaiseksi kuin se oli ennen asettamista, toisin sanoen `setVisible(setVisible(k)) == k`.

3.2.3 Formula

Formula sisältää sen valinnasta seuraavien todistusalipuiden lisäksi kyseisen kaavan jäsennetyn muodon (ParsedFormula). Formula toteuttaa Javan kloonirajapinnan Cloneable.

```
new(ParsedFormula f)
```

Konstruktori luo uuden kaavan ja kiinnittää sille jäsennyspuun. Jäsennyspuutonta kaavaa ei ole olemassa, joten sellaisen luomiseen ei tule antaa konstruktoria.

```
addChildren(int rule, Proof children[]);
```

AddChildren lisää todistukseen jatkon sääntönumerolla *rule*.

```
clone(void)
```

Clone palauttaa ohuen kopion kaavasta, toisin sanoen kopion jossa Formulan osaluokkien instansseja ei ole kopioitu, vaan ne ovat viitteitä vanhoihin. Lisäksi clone poistaa mahdolliset olemassaolevat alitodistukset luodusta kopiosta.

3.2.4 ParsedFormula

ParsedFormula esittää kaavan jäsennyspuuta ja on siten rekursiivisesti määritelty tietotyyppi (binääripuu). ParsedFormula toteuttaa Javan kloonirajapinnan Cloneable ja sisältää seuraavat metodit.

```
Parse(Stream s)
```

Parse jäsentää tietovirrasta kaavan aiemmin kuvatussa muodossa. Jäsentäjä ei kiinnitä operaattoreiden nimiä vaan sen sijaan erottelee atomit ja konnektiivit pienen ja suuren alkukirjaimen (tai jos myöhemmin nähdään tarpeelliseksi, jonkin tarkennetun erityispiirteen) perusteella. *Parse* tallentaa jokaiseen solmuun merkkijonon (String), joka syötteessä sillä paikalla oli, ja jättää niiden mahdollisen korvaamisen tietokoneelle helpommin käsiteltävissä olevilla olioilla (Object) kalkyylille (Calculus).

```
clone(void)
```

Clone palauttaa syvän (rekursiivisen) kopion kaavasta.

3.2.5 ProofParser

ProofParser luokkien tehtävä on luoda kokonaisia todistuspuita parametrina saadun tietovirran perusteella. Rajapintaan kuuluu, että SequentParser on osa ProofParseria, mutta toteutuksen ei ole toki pakko hyödyntää sitä. Luontioperaatiolle on käänteisoperaatio, ja luokkien jotka toteuttavat sekä luonnin että tallennuksen tulisi varmistaa että tallennettu tieto on luettavissa siten että luetun perusteella muodostettu todistuspuu on samanlainen kuin se, jonka perusteella tallennusoperaatio suoritettiin. Toinen näistä operaatioista voidaan jättää toteutuksissa toteuttamatta, jolloin kyseisen operaation tulisi heittää poikkeus ParserOperationNotImplementedError.

ProofParser on rajapinta (interface), jonka varsinaiset toteutukset perivät.

ProofParser sisältää seuraavat metodit.

```
Proof parseProof(Stream s)
writeToStream(Stream s, Proof p)
```

ParseProof on yllä kuvattu jäsennusoperaatio, ja *writeToStream* sitä vastaava tallennusoperaatio.

3.2.6 SequentParser

SequentParser luokkien tehtävä on luoda tietovirran perusteella kaksiulotteisia taulukoita jotka sisältävät kaavoja (Formula). Kääntäen, se tulostaa kaksiulotteisen Formula-taulukon perusteella tietovirtaan jotain, jonka mainittu käänteinen operaatio osaa lukea siten että tuloksena on samanlainen taulukko. Toinen näistä operaatioista voidaan jättää toteutuksissa toteuttamatta, jolloin kyseisen operaation tulisi heittää poikkeus ParserOperationNotImplementedError.

SequentParser on rajapinta (interface), jonka varsinaiset toteutukset perivät.

SequentParser sisältää seuraavat metodit.

```
ParsedFormula[][] parseSequent(Stream s)
writeToStream(Stream s, ParsedFormula[][] seq)
```

Kyseisten metodien toiminta on täysin analoginen vastaavien ProofParser metodien kanssa.

3.3 Käyttöliittymä

Ohjelma käyttää käyttöliittymäkomponentteja Javan AWT-kirjastosta. Käyttöliittymä on jaettu osiin periyttämällä luokista uusia aliluokkia, johon sisällytetään tämän luokan toiminnot toteuttavat rutiinit. Samat asiat olisi voitu tehdä myös ilman periytystä, mutta käyttöliittymäosa jakautuu loogisesti eri tavoin toimiviin osiin, jotka voidaan toteuttaa itsenäisesti ja jotka riippuvat periytetystä luokasta.

Ohjelmassa on kaksi Frame-tyyppistä ikkunaa, pääikkuna ja navigointi-ikkuna, sekä useampia Dialog-tyyppisiä ikkunoita: tulostusikkuna, lataus-, tallennus- ja uusi-ikkunat sekä asetusikkuna. Lataus-, tallennus- ja asetusikkunoiden toiminta toteutetaan pääikkunassa, muut on periytetty omiksi luokikseen.

Kaksi luokkaa perii Canvas-piirtoalueluokan: pääikkunan todistuksenpiirtoalue sekä navigointi-ikkunan puunpiirtoalue.

3.3.1 Käynnistysluokka ProEdit

Ohjelma käynnistetään ajamalla tämän luokan main-metodi. Luokka sisältää useamman todistusikkunan hallintakomponentit.

```
// Lista ohjelman käyttämistä todistusikkunoista, sisältää aina
// vähintään yhden ikkunan tai muuten ohjelma lopetetaan.
Vector mainWindows // (of MainWindow), != null

public static void main()

// Avaa tyhjän ikkunan, tiedoston nimeksi name esim. "Untitled 1".
void newWindow(String name)

// Avaa ikkunan, johon todistuksen
void newWindow(Calculus, String name)

// Sulkee yhden ikkunan, ja ohjelman jos ei yhtään jäljellä
void closeWindow(MainWindow)

// Sulkee kaikki ikkunat ja ohjelman
void exit()
```

3.3.2 Pääikkuna MainWindow

Pääikkunaluokka toteuttaa pääikkunan ja navigointi-ikkunan tapahtumankäsittelyn (poislukien Canvas-osat) ja rutiinit muiden ikkunoiden näyttämiseen, jolloin toiminta ohjataan näille ikkunoille. Pääikkunassa on tallennus- ja latausrutiinit.

```
ProEd program       // != null
Dialog saveDialog   // != null
Dialog loadDialog   // != null
Frame naviWindow    // != null
NaviArea naviArea   // != null, saadaan proofArea-oliolta.
ProofArea proofArea // != null

public void actionPerformed(Event)
```

```
// Konstruktori, ProEd != null
MainWindow(ProEd, Calculus)

// Avaa uusi todistus -ikkunan
void preNew()

// Avaa uuden todistuksen tähän ikkunaan, jos ikkunassa ei ole
// todistusta, muuten tekee ProEdit-luokalla uuden ikkunan.
void new(Calculus)

// Avaa tallennusikkunan
void preSave()

// Tallentaa todistuksen, kutsuu Calculus-olion save-metodia
void save(String filename)

// Avaa latausikkunan
void preLoad()

// Avaa todistuksen tähän ikkunaan, jos ikkunassa ei ole vielä
// todistusta, muuten tekee ProEdit-luokalla uuden ikkunan.
void load(String filename)

// Avaa vientidialogin
void preExport(Proof)

// Joko vaihtaa navi-ikkunan tilaa suljetusta näkyvään (c=false)
// tai pakottaa sen kiinni (c=true)
void naviToggle(boolean c)

Käsiteltävät tapahtumat:
- Menu-New: kutsu preNew()
- Menu-Load: kutsu preLoad()
- Menu-Save: kutsu preSave()
- Menu-Export: kutsu preExport()
- Menu-Close, WindowClosed: kutsu ProEdit-olion closeWindow()-metodia
- Menu-Navi Toggle: kutsu naviToggle()
- Menu-Exit: kutsu ProEdit-olion exit()
- Vastaavat toiminnot ikkunassa oleville nappuloille
- Navi-ikkunan WindowClosed: kutsu naviToggle()
```

3.3.3 NewDialog

Luokka sisältää Uusi todistus -dialogin käyttämät toiminnot, ja se myös ohjaa käyttäjän merkkijonona antaman sekventin jäsentäjälle. Jos jäsennys onnistui, sekventti annetaan eteenpäin pääikkunalle.

```
MainWindow mainWindow // != null

// Konstruktori, MainWindow != null
NewDialog(MainWindow)

public void actionPerformed(Event)

// Jäsentää Sequent-luokan avustuksella merkkijonon
Sequent parse(String)

Käsiteltävät tapahtumat:
- Ok-nappulaa painettu. parse-metodista saadulla Sequent-oliolla luodaan
  ikkunan valintojen perusteella uusi Calculus-olio jolla kutsutaan
  pääikkunan new-metodia, ja ikkuna suljetaan.
- Cancel-nappulaa painettu, WindowClosed. Ikkuna suljetaan.
```

3.3.4 PrintDialog

Tulostusikkunassa on tulostuksen ja viennin toteuttavat rutiinit. Ikkuna sisältää myös molemmat piirtoalueluokat, navigointialue toimii tässä tapauksessa tulostusalueen valintaikkunana ja todistusalue esikatseluruutuna.

```
MainWindow mainWindow  // != null
Calculus proof         // != null
ProofArea previewArea  // != null
NaviArea selectionArea // != null

// Konstruktori, Calculus != null
PrintDialog(MainWindow, Calculus)

// Lukee selectionArea:n valinnat ja muut kenttien tiedot ja
// tulostaa Calculuksen näiden perusteella.
void export()

Käsiteltävät tapahtumat:
- Export-nappulaa painettu. Kutsu export-metodia
- Close-nappulaa painettu, WindowClosed. Sulkee ikkunan.
```

3.3.5 ProofArea

Asettelee ja piirtää todistukset ruudulle.

ProofArea ja NaviArea ovat tiiviisti toistensa kanssa yhteistyössä (niiden välillä on 1-1 -suhde), joten ProofAreaa luotaessa luodaan sille myös NaviArea. NaviAreaa ei luoda missään muualla (luomista ei pystytä kuitenkaan pakolla estämään). NaviArea-olio pyydetään tältä oliolta ja sijoitellaan sitten sopivaan paikkaan käyttöliittymässä.

```
// Käytetty calculus, jos on null, mikään metodi ei tee mitään
// (tapahtuu silloin kun ohjelma on juuri avattu)
Calculus calculus

// Viittaus puun juureen. ProofArea käyttää Proof-puuta suoraan.
// Sama kuin calculus.root.
Proof root

// Piirtoalueen keskipisteessä olevan pisteen koordinaatit
// puun virtuaalisen sijoittelun suhteen, jossa puun juuri
// on koordinaateissa 0,0.
int centerX, centerY

// Piirtoalueen keskipisteessä oleva kaava. Jos != null,
// centerXY-tiedot lasketaan tämän avulla.
Proof centerProof

// Vastaava naviArea, kutsutaan applyn jälkeen tämän paint-metodia
// jotta sen ruutu päivittyy. Luodaan konstruktorissa, joten
// sen jälkeen aina != null.
NaviArea naviArea


// Kutsutaan aina hiirtä liikutettaessa. Voidaan muuttaa hiiren
// alla olevan tekstin väriä.
public void mouseMotionListener(MouseEvent)

// Kutsutaan hiirtä klikatessa. Selvitetään paikka, siihen piirretty
// Proof-olio, olion kaava jota klikattiin, ja viimeisenä kaavassa oleva
// konnektiivi. Näiden perusteella kutsutaan calculus-olion apply-metodia.
public void mouseListener(MouseEvent)

// Asettelee Proof-oliot alueelle puun muotoon ja piirtää sen
public void paint(Graphics g)

// Keskittää alueen koordinaatteihin
void center(int x, int y)

// Keskittää alueen kaavaan
void center(Proof)

// Näyttää vain parametrien perusteella lasketun osan puusta
void preview(Proof root, Vector leaves)
```

3.3.6 NaviArea

Asettelee ja piirtää navigointipuun ruudulle.

```
ProofArea proofArea // != null
Proof root // == proofArea.root
Proof exportRoot
Vector exportLeaves // (of Proof)

// Konstruktori, ProofArea != null
NaviArea(ProofArea)

// Kutsutaan aina hiirtä liikutettaessa. Voidaan muuttaa hiiren
// alla olevan symbolin väriä.
public void mouseMotionListener(MouseEvent)

// Kutsutaan hiirtä klikatessa.
// Navi-moodissa selvitetään hiiren paikka, siihen piirretty
// Proof-olio ja kutsutaan proofArean center-metodia.
// Export-moodissa merkitään tulostusalue ja kutsutaan proofArean
// preview-metodia.
public void mouseListener(MouseEvent)

// Asettelee Proof-oliot puuksi, selvittää värityksen (riippuen
// moodista) ja piirtää oliot.
public void paint(Graphics g)
```

4 Käyttötapausten toteutus

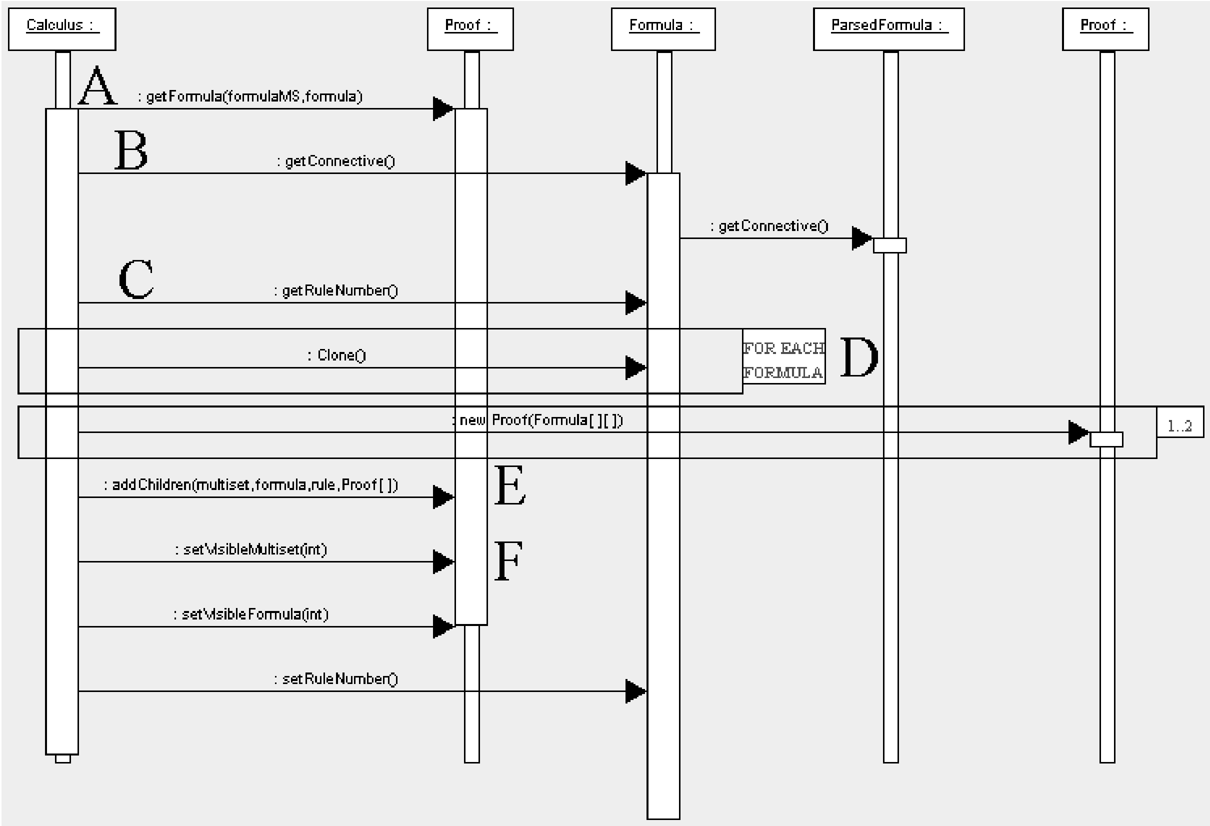
Käyttötapausten toteutusta tutkitaan sekvenssikaavioiden avulla. Tässä rajoitutaan tutkimaan niitä käyttötapauksia, jotka käyttävät tietorakenneluokkien palveluja. Näitä ovat kaavanosan klikkaaminen (apply), tietovirran lukeminen (read), tietovirtaan kirjoittaminen (write) ja viimeisimmän toiminnon peruuttaminen (undo).

Erityistä huomioitavaa sekvenssikaavioissa on se, että niissä käytetään selkeyden vuoksi arvoja asettavia ja palauttavia metodeja (aksessorit), vaikka niitä ei oikeasti toteutetakaan.

4.1 apply

Kun käyttäjä klikkaa todistuksen tiettyä kaavaa, käyttöliittymärakenne kutsuu Calculus-luokan metodia apply, joka soveltaa kalkyyliä kaavaan. Tätä ennen käyttöliittymä on selvitettävä koordinaattien avulla mistä todistuksen osasta, sekvenssistä ja kaavasta on kyse.

Kuvan 4 kaaviossa nähdään vaiheissa A-G kuinka apply-metodi suoritetaan. Ensimmäiseksi kalkyylin täytyy selvittää, mitä sääntöä sen tulee soveltaa. Kalkyyli saa Proof-oliolta viitauksen klikattuun kaavaan (vaihe A). Kaavalta kysellään sen konnektiivia ja sääntönumeroa (vaiheet B ja C). Näiden avulla tiedetään sovellettava sääntö.



Kuva 2: Sekvenssikaavio: apply

Seuraavaksi kloonataan Proof-olion kaikki kaavat (vaihe D), koska nämä kopioidaan lapsisolmuihin. Myös klikattu Formula kloonataan, mutta tältä otetaankin vasen ja oikea alipuu lapsisolmuihin. Tätä operaatiota ei näy kaaviossa 4.

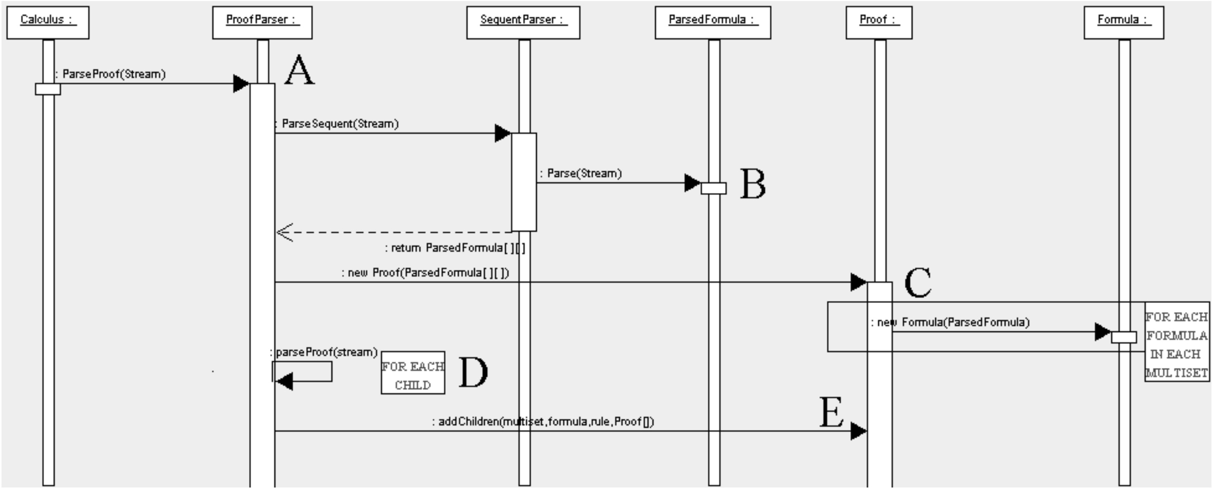
Lopuksi palvelulla createChildren() luodaan todistuspuun isäsolmulle viittaukset uusiin lapsisolmuihin (vaihe E). Isäsolmun kaavojen näkyvyydet päivitetään (vaihe F) ja lopuksi klikattuun formulaan merkataan sääntönumerolla mitä sääntöä sovellettiin (vaihe G).

4.2 read

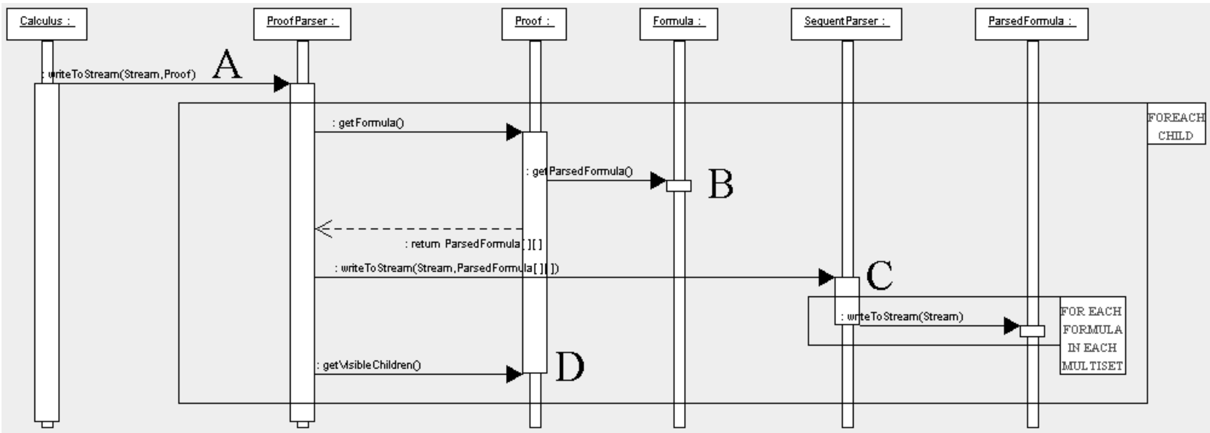
Ohjelma lukee tietovirtaa silloin, kun käyttäjä syöttää uuden kaavan tai lataa tiedostosta vanhan todistuksen. Kummassakin tapauksessa toiminta on samanlaista. Kuvan 4 kaavion vaiheissa A-E nähdään, kuinka tietovirta parsitaan todistuspuuksi. Tietovirran tarkempi esitysmuoto löytyy kappaleesta 2.1.1.

Aluksi Kalkyyli pyytää ProofParseria parsimaan tietovirran (vaihe A). ParseProof-metodin tarkempi toiminta nähdään ao. seutoJavapätkästä.

```
public static Proof parseProof(Stream stream) {
    int multiset = read_multiset_from(stream);
    int formula  = read_formula_from(stream);
```



Kuva 3: Sekvenssikaavio: read



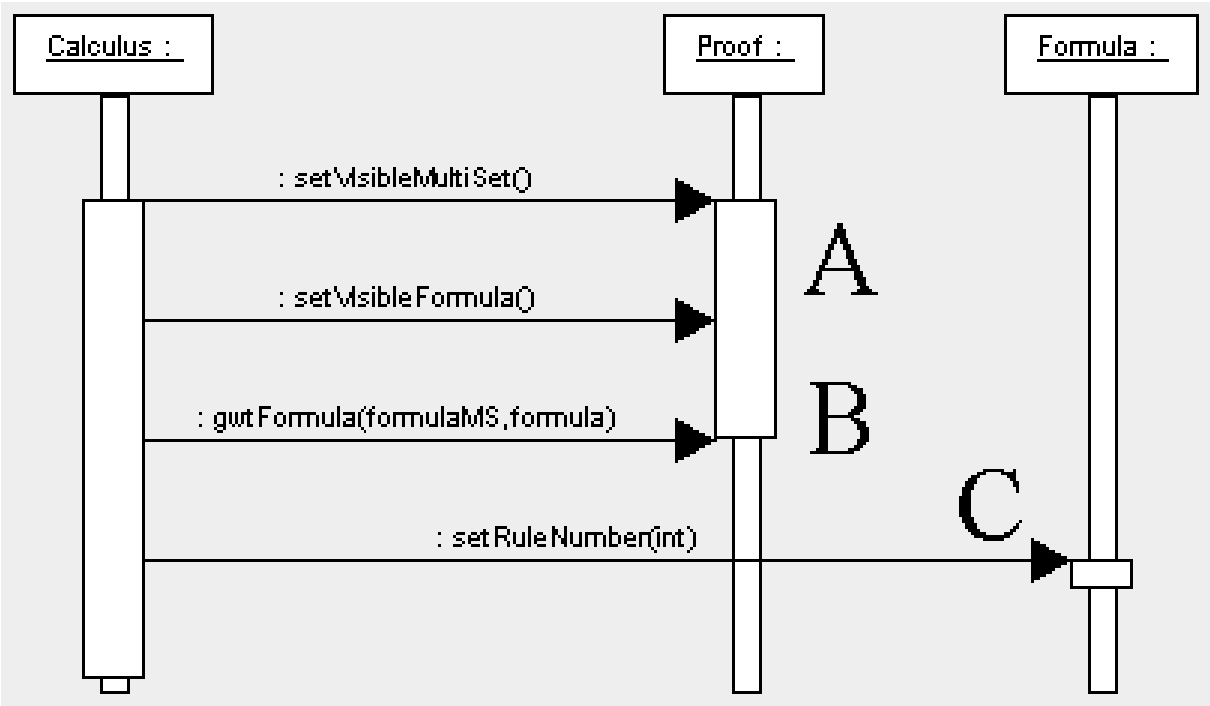
Kuva 4: Sekvenssikaavio: write

```
    ParsedFormula[][] pf = SequentParser.ParseSequent(stream); //B
    Proof p = new Proof(pf); //C

    Proof child  = new Proof[2];
    child[LEFT]  = parseProof(left_part_of_stream);  //D
    child[RIGHT] = parseProof(right_part_of_stream);

    p.addChildren(multiset,formula,rule,child); //E
}
```

Koodissa näkyvät samat vaiheet B-D kuin kuvassa 4. Ensiksi luetaan tietovirrasta, mistä sekvenssistä ja kaavasta todistus jatkuu eteenpäin. Seuraavaksi tietovirta sysätään sekvenssiparsijalle, joka puolestaan pyytää Parsedformulaa parsimaan sen (vaihe B). Paluuarvona saadaan ParsedFormula-taulukko, jossa ensimmäinen indeksi kertoo sekvenssin



Kuva 5: Sekvenssikaavio: undo

numeron ja toinen indeksi formulan numeron.

Uusi todistus ja sen kaikki formulat luodaan saadun taulukon perusteella (vaihe C). Tämän jälkeen tietovirta jaetaan kahteen osaan ja kutsutaan rekursiivisesti parseProofia, joka luo lapsisolmut (vaihe D). Lopuksi isäsolmuun p luodaan viittaukset lapsiin (vaihe E).

4.3 write

Tietovirtaan kirjoitetaan silloin, kun käyttäjä haluaa tallettaa todistuksen tai tulostaa sen. Kuvassa 4 nähdään vaiheissa A-D kuinka todistuspuu kirjoitetaan tietovirtaan.

Käyttötapaus alkaa siitä, kun kalkyyli pyytää ProofParseria kirjoittamaan virtaan todistuspuun (vaihe A). ProofParser pyytää Proofilta todistuksen kaavat taulukkona (Vaihe B). ProofParser kutsuu SequentParseria joka osaa tulkita taulukkoa ParsedFormulan avulla ja kirjoittaa kaavat tietovirtaan (vaihe C). Lopuksi Proofparser selvittää saamansa todistuksen lapsisolmut (Vaihe D) ja suorittaa vaiheet B-D rekursiivisesti kullekin lapsisolmulle.

4.4 undo

Undo peruuttaa viimeisimmän applyn. Käytännössä todistuspuun oksien näkyvyyksiä muutetaan siten, että uusi oksa katoaa näkyvistä ja vanha palautetaan näkyviin. Kuvassa 4 nähdään, kuinka se vaihe vaiheelta tapahtuu.

Kalkyyliin on talletettu tiedot todistuksen vanhat näkyvyydet. Kun Undo käynnistyy, kalkyyli asettaa Proofille vanhat talletetut näkyvyysarvot ja saa paluuarvona uudet näkyvyydet jotka se tallettaa kenttiinsä (Vaihe A). Tästä syystä undon undo toimii kuten redo. Seuraavaksi Kalkyyli pyytää todistukselta sitä kaavaa, johon kalkyyliä on sovellettu (vaihe B) ja kasvattaa kaavan sääntönumeroa (vaihe C), jos sovellettavia sääntöjä oli kaavalle useita.

5 Käyttöliittymä

Ohjelman käyttöliittymän perusyksikkö on ikkunapari, joista toinen, pääikkuna (luokka MainWindow), sisältää käsillä olevan todistuksen, valikot sekä pikapainikkeet, ja toinen pelkästään navigointipuun. Navigointipuuikkuna voi olla myös piilossa ja sen saa takaisin esille valikosta.

Avoinna voi olla kerrallaan useita ikkunapareja. Yhden ikkunaparin tunnistaa toisiinsa liittyväksi ikkunan otsikoista, joissa näkyy sama tiedostonimi. Avoinna on aina vähintään yksi ikkunapari. Molempien ikkunoiden kokoa voi muuttaa. Tällöin todistusalueen ja navigointialueen koko muuttuu nappuloiden pysyessä ennallaan.

5.1 Valikot ja pikapainikkeet

Pääikkunassa on kolme valikkoa, File, Edit ja Help. File-valikko sisältää seuraavat toiminnot:

- New... - Avaa ikkunan, johon kirjoitetaan uusi todistettava kaava.
- Load... - Avaa standardin tiedostonlatausdialogin, jolla valittu tiedosto ladataan näkyviin.
- Save - Jos tiedostolle ei ole asetettu nimeä, toimii kuten Save as..., muuten tallennetaan todistus tiedostoon.
- Save as... - pyydetään standardilla dialogilla tiedostonimi, johon todistus tallennetaan.
- Export... - Avaa vientidialogin, jolla todistus voidaan tulostaa.
- Close - Sulkee avoinna olevan ikkunan ja jos ikkuna oli viimeinen, sulkee koko ohjelman. Varmistusdialogi ennen sulkemista.
- Exit - Sulkee kaikki ikkunat ja ohjelman. Varmistusdialogi ennen tätä.

Seuraavana on pieni Edit-valikko, jossa on seuraavat:

- Undo - Toteuttaa undo-toiminnon. Toimii kuten allakuvattu Undo-nappula eli muuttuu myös Redo-tekstiksi.

- Hide navigation window - Piilottaa navigointi-ikkunan, valikkoon tulee tällöin checked-valinta. Navigointi-ikkuna voidaan piilottaa myös sulkemalla se ikkunan "ruksista", jolloin checked-valinta syttyy myös. Tällöin valitseminen palauttaa ikkunan ruudulle ja poistaa valinnan.
- Large font - Valitsee suuremman fontin todistusalueen kaavoille, jolloin valikkoon tulee checked-valinta. Uudelleen valitseminen palauttaa pienen fontin ja poistaa checked-valinnan.

Help-valikko sisältää vain About... -valinnan joka avaa ohjelmasta kertovan dialogi-ikkunan.

Pääikkunan vasen puolisko on varattu todistusalueelle (luokka ProofArea), oikealle puolelle on sijoiteltu pikapainikkeet: kaikki file-valikon painikkeet paitsi ehkä Hide navigation window, Close ja Exit, ja lisäksi Undo-nappula. Undo-nappulan teksti muuttuu Redo:ksi kun Undo-toiminto on suoritettu, ja palaa takaisin Undoksi jos Redo-nappulaa painettiin tai tehtiin todistukselle jokin muutos.

5.2 Todistusalue

Pääikkunan vasen puoli koostuu todistusalueesta, johon itse todistuspuu piirretään. Alueen keskipiste on keskitetty johonkin todistuspuun osaan. Kun alueen koko muuttuu (ikkunan kokoa muutettaessa) tämä keskitys säilyy ennallaan. Todistusalueen reunoilla on myös vierityspalkit, joilla keskityskohtaa voidaan muuttaa.

Toim. huom: Paitsi että tämä on mahdotonta, koska vierityspalkkien käyttö voi siirtää ruudun keskikohdan todistuspuun ulkopuolelle. Vierityspalkit operoivat yleensä käyttöliittymissä jatkuvassa ja lineaarisessa avaruudessa. Todistuspuu osineen muodostaa hierarkkisen ja diskreetin avaruuden. Näiden yhdistäminen lienee haastavaa.

Todistuspuun yksi solmu koostuu sekventistä ja siihen mahdollisesti sovelletusta säännöstä. Solmun lapset ovat tästä solmusta valitulla säännöllä johdetut uudet sekventit (joita voi olla käyttöliittymän nykytoteutuksessa enintään kaksi).

Sekventistä väritetään eri värillä, vaikkapa sinisellä, (tai ympyröidään), se konnektiivi johon sääntöä on sovellettu. Jos käyttäjä on klikannut suluissa sisempänä olevaa konnektiivia, tätä valintaa ei näytetä itse siinä kaavassa jossa klikkaus on tehty, vaan tässäkin kaavassa uloin konnektiivi on merkitty. Koko kaava johon sääntöä sovellettiin voidaan värittää jollain toisella värillä (eri sävyinen sininen esimerkiksi).

Todistuksen valmistuminen kokonaan voidaan ilmaista esimerkiksi värittämällä koko puu jollain toisella värillä tai avaamalla pieni asiasta kertova dialogi. Tämä näkyy kuitenkin myös navigointi-ikkunasta, joten se ei ole välttämättä tarpeellista.

Käyttäjä voi klikata mitä tahansa ruudulla näkyvää konnektiivia, jolloin todistusta puretaan auki. Propositioiden tai sekventtinuolen klikkaaminen keskittää ruudun tähän sekventtiin, ja näin tapahtuu myös konnektiivia klikatessa. Mitään todistuspuussa näkyvää haaraa ei voida piilottaa näkyvistä muuten kuin valitsemalla haaran alkukaavasta jokin toinen sääntö sovellettavaksi.

Jos toteutusvaiheessa löydetään helppo tapa (toteutukseltaan ja käytännöllisyydeltään) avata jokin ruudulla näkyvä sekventti omaan ikkunaansa alimmaksi sekventiksi, sellainen toiminto voidaan toteuttaa.

5.3 Navigointipuu

Navigointipuu on omassa ikkunassaan. Jos puu jatkuu ikkunan ulkopuolelle, siinä on vierityspalkit.

Navigointipuun solmu vastaa yhtä sekventtiä todistusalueessa ja sen nimenä on joko sovelletun säännön nimi, tai kysymysmerkki, jos sekventtiin ei ole vielä sovellettu mitään. Solmujen välillä on pieni viiva.

Solmut on väritetty. Musta solmu vastaa sekventtiä joka on kesken ja keskellä puuta. Punainen solmu vastaa sekventtiä, joka on kesken ja puun lehti (eli kysymysmerkki). Vihreällä värillä on merkitty kaikki ne solmut, jotka ovat valmiita, eli kaikki niiden lehdet on suljettu. Kun koko puu on vihreä, todistus on valmis. Punaisella näkyvät solmut ovat kesken ja niitä pitää vielä jatkaa.

Solmun klikkaaminen keskittää todistusalueen tähän sekventtiin. Muualla puun alueella klikkaaminen ei tee mitään. Aina kun todistusalue keskitetään johonkin kaavaan, merkitään esimerkiksi laatikolla navigointipuun se solmu, johon keskitys on tehty. Jos todistusalueen kohdistusta muutetaan vierityspalkeilla, tämä merkki ei enää ole täsmällinen joten joko poistetaan merkki näkyvistä kokonaan tai dynaamisesti lasketaan ruudun keskustaa lähinnä oleva sekventti ja merkitään tämä navigointipuussa, tai keksitään jokin muu tapa merkitä navigointipuusta se kohta jota todistusikkunassa ollaan tarkastelemassa.

5.4 Uusi todistus -ikkuna

Ylimpänä uusi todistus -ikkunassa on kalkyylin valitseva radiobutton, tässä versiossa siis vain valinnat G3IP ja G3CP.

Sen lisäksi siinä on kentät sekventin syöttöä varten. Tämä voidaan toteuttaa joko yhdellä kentällä (jolloin käyttäjä kirjoittaa kenttään itse sekvenssinuolen ja erottelee eri kaavat toisistaan pilkuilla), kahdella kentällä, joiden välissä on sekvenssinuoli, tai usealla kentällä, joihin kuhunkin kirjoitetaan yksi kaava ja joista puolet on nuolen toisella ja puolet toisella puolella. Toteutuksessa valitaan parhaiten toimiva ratkaisu.

Lisäksi ikkunassa on alue, jossa näkyvät eri kirjoitusmuodot eri matemaattisille symboleille. Alueen sisältö voisi muuttua kun radiobuttonia vaihdetaan, jos käytetyt merkinnät riippuisivat kalkyylistä, mutta koska nyt toteutettavat kalkyylit käyttävät samaa merkintätapaa, ei muuttumista tehdä. Radiobuttonin vaihtaminen ei myöskään tee mitään tekstikentille, vaan vasta kun Ok-nappulaa on painettu, kenttien sisällöt tarkistetaan ja virheen sattuessa huomautetaan käyttäjää.

Lisäksi ikkunan alalaidassa ovat Ok- ja Cancel-nappulat, joista jälkimmäinen kumoaa todistuksen luonnin ja edellinen tarkistaa kaavan syntaktisen oikeellisuuden ja avaa uuden todistusikkunan sille.

5.5 Vienti-ikkuna

Vienti-ikkunassa on radiobutton-valinta sille, missä muodossa vienti halutaan tehdä. Tässä versiossa valittavana ovat ASCII- ja LaTeX`proof.sty`-vaihtoehdot.

Seuraavana on tiedoston valinta. Tämä voidaan valita esimerkiksi kirjoittamalla tekstikenttään tiedoston nimi tai putkikomento (esim. "`|lpr`"). Vaihtoehtoisesti voidaan valita tiedosto "Browse..."-napilla joka avaa standardin tiedostontallennusdialogin, jolloin valitun tiedoston nimi ilmestyy tekstikenttään. Jos `proof.sty`-tulostus halutaan putkittaa suoraan tulostimelle, käyttäjän on tehtävä itse komennot jotka lisäävät tiedostoon tarvittavat otsikkotiedot, tulkkaavat LaTeX:illa ja `dvips`illä ja syöttävät tämän kirjoittimelle.

Ikkunassa on kaksi pääosaa, jotka muistuttavat pääikkunan ja navigointi-ikkunan piirtoalueita. Pääikkunaa muistuttava osa toimii esikatseluna, joka näyttää dynaamisesti käyttäjälle sen alueen, jonka tulostuskomento sillä hetkellä tulostaisi.

Navigointi-ikkunaa muistuttavasta alueesta valitaan tulostettava alue. Molempien alueiden vierellä on tässä vierityspalkit. Alueen lähellä, esim. alapuolella, on ohjepainike, joka avaa pienen ohjeruudun siitä, kuinka tulostusalue valitaan.

Ikkunassa on lisäksi "Export"- ja "Close"-nappulat, joista edellinen tulostaa valitun alueen tiedostoon sekä pyyhkii navigointi-ikkunan valinnat ja jälkimmäinen sulkee ikkunan.

Tulostusalue valitaan seuraavasti: Käyttäjä käyttää kahdenlaisia hiiren klikkauksia, esimerkiksi tavallinen klikkaus ja shift-klikkaus. Tavallinen klikkaus valitsee tulostusalueen alimman solmun. Shift-klikkaus rajaa aluetta ylhäältä päin siten, että klikattua solmua ylemmät solmut jätetään pois valitusta alueesta. Klikkaus pyyhkii aina aiemmin valitut shift-klikkaukset, mutta shift-klikkaukset toimivat toggle-periaatteella.

Toim. huom: edelleen jää epäselväksi rajataanko ylhäältä vain kyseisen haaran vai myös muiden haarojen korkeammalle nousevat solmut.

Alueen valintametodia voidaan muuttaa, jos toteutuksen aikana havaitaan vielä parempi ratkaisu. Tähän mennessä valintametodi on muuttunut niin monta kertaa, että ei ole mitään takeita, että edellä esitetty metodi olisi absoluuttisesti paras. Metodin muuttaminen kesken kaiken ei tuottane suuria toteutusongelmia.

Toim. huom: Aamen.

Esikatseluruudussa voitaisiin näyttää tulostettava alue mustana tekstinä valkoisella pohjalla, ja tulostamatta jäävät kaavat harmaalla tekstillä. Alueenvalintaruudussa näkyy valittu tulostusalue jollain näkyvällä värillä, esimerkiksi vihreällä.

Kun jokin alue on tulostettu onnistuneesti ikkunassa, alueenvalintaruutu palaa tilaan, jossa kaikki solmut on valittu. Esikatseluruutu näyttää tämän myös. Alueenvalintaruudussa ne solmut jotka tulostettiin muuttuvat kuitenkin "varjostetuiksi", mistä käyttäjä näkee heti, mitä kaavoja ei tarvitse enää tulostaa. Varjostus näkyy niin, että normaalisti mustatekstinen solmun nimi näkyy harmaana, tai jos solmu on valittu tulostettavaksi ja näkyy vihreänä, se näkyy tällöin vaaleamman vihreänä. Lisätulostaminen lisää varjostettujen solmujen määrää, ja varjostukset katoavat vain kun ikkunan sulkee ja avaa uudelleen.

Toim. huom: Puheet tulostamisesta ja viennistä kummittelevat aika sitkeästi. Ehkä pitäisi sittenkin käyttää kaikkialla termejä tulostaminen ja tiedostoon tulostaminen.

5.6 Virhetilanteiden hallinta

Ohjelman toiminnan aikana ei voi tapahtua kovin monenlaisia virheitä. Tässä vaiheessa näyttää siltä että vain kolmenlaisia virheitä voi tulla, mutta toteutusvaiheessa saattaa löytyä joitain uusia (jokin käyttöliittymäkomponentti heittää poikkeuksen, johon ei voida varautua), mutta nämä käsitellään vastaavalla tavalla.

- Tiedostojärjestelmän virheet: lataus/tallennus/export

- Käytetty kaava on vääränmuotoinen

- Ladattava tiedosto on vääränlainen

- Kriittiset virheet kuten pinon tai kasan loppuminen.

6 Testaussuunnitelma

Testaussuunnitelmassa kerrotaan testauksen yleispiirteistä ja käytettävistä testausmenetelmistä. Testauksessa on tarkoitus löytää virheistä suurin osa. Testauksesta kirjoitetaan testausdokumentti jossa raportoidaan kaikki testit ja tiedot niiden onnistumisesta.

6.1 Yleistä testauksesta

Ohjelman testaus jakaantuu kolmeen osaan, Java-luokkien yksikkötestaukseen, testattujen luokkien yhteistoimintaa testaavaan integrointitestaukseen sekä käyttötapauksiin perustuvaan järjestelmätestaukseen. Yksikkötestauksessa etsitään lähinnä erillisten moduuleiden ohjelmointivirheitä ja niiden sisäisessä toiminnallisuudessa olevia vikoja, kun taas integrointitestauksessa testataan useammasta moduulista koostuvia kokonaisuuksia. Validointitestauksessa ohjelmiston toiminnallisuus testataan suorittamalla järjestelmällisesti mahdollisimman kattavasti kaikki käyttötapaukset.

Projektissa noudatetaan XP-mallin mukaisesti testausta jo ohjelmoinnin yhteydessä, joten testaus alkaa samaan aikaan kuin toteutuskin. Toteutusvaiheen jälkeen on varattu kolme viikkoa aikaa pelkkään testaukseen. Testauksen deadline on 6.5, jolloin myös testausdokumentti on valmis.

Testaukseen osallistuvat kaikki ryhmän jäsenet.

6.2 Testausmenetelmät

Testitapauksia valittaessa on käytettävissä kaksi päästrategiaa, mustalaatikkotestaus (black box testing, functional testing) tai lasilaatikkotestaus (glass/white box testing, structural testing).

White box- testauksessa tunnetaan kohteen sisäinen rakenne ja toteutus Lasilaatikkotestauksessa testidatan valinnalla pyritään siihen, että testattava ohjelma tulee testattua mahdollisimman kattavasti.

Black box- testauksessa syötteet jaetaan ekvivalenssiluokkiin, joista samaan kuuluvat ne jotka käyttäytyvät samalla lailla testauksen kannalta. Näin ollen mahdollinen virhe löytyy jokaisella tämän luokan arvolla.

6.3 Yksikkötestaus

Toteutuksen jälkeen on jätetty kolme viikkoa pelkkään testaukseen, mutta jo toteutusvaiheessa ohjelmoija testailee tekemäänsä koodia. Testiluokka kirjoitetaan ennen varsinaisen luokan kirjoittamista, jolla luokan valmistuttua sitä voidaan testata. Testit tehdään whitebox testauksena ja niissä pyritään saavuttamaan lausekattavuus, eli jokaisessa ohjelman lauseessa pyritään käymään vähintään kerran.

Jotta voidaan olla varmoja että testausluokka on tehty ennen varsinaista luokkaa, tämä laitetaan CVS:ään heti testauksen jälkeen.

6.4 Integrointitestaus

Integrointitestauksessa pyritään testaamaan järjestelmän eri osien toimivuus yhdessä. Se tehdään siinä vaiheessa kun yksikkötestaus liitettävien luokkien osalta on valmis. Testaus tehdään lasilaatikkotestauksena. Erityisesti luokkien väliset rajapinnat ovat huomion kohteena. Integrointitestauksessa käytetään bottom up- menetelmää, luokkien toimintaa yhdessä testaillaan kun luokat on valmistuneet. Jos jotain integrointitestauksessa mukana ollutta luokkaa muokataan, joudutaan testit tekemään uudestaan.

Integrointitestaukseen osallistuvat luokkien tekijät.

6.5 Validointitestaus

Validointitestauksessa pyritään todentamaan että ohjelmisto toimii määritellyllä tavalla. Siinä pyritään selvittämään ulkoisesti havaittava toiminta, suorituskyky ja virheensietokyky. Määrittelydokumentissa esitettyjen käyttötapausten avulla kehitetään sopivat testit. Validointitestaukseen osallistuvat kaikki ryhmän jäsenet.

Testitapauksia esitellään seuraavassa.

Todistuksen lataaminen tiedostosta:

Testitapaus	Odotettu tulos
Ladataan kelvollinen todistus	Onnistuu, todistus latautuu.
Ladataan kelpaamaton tiedosto	Lataus ei onnistu.
	Ohjelma ilmoittaa virheestä ja pysytään latausikkunassa.

Todistuksen tallentaminen:

Testitapaus	Odotettu tulos
Tallennetaan todistus levylle.	Todistus tallentuu.

Todistuksen tulostaminen:

Testitapaus	Odotettu tulos
Tulostetaan koko puu.	Toimii.
Tulostetaan puusta vain osa.	Haluttu osa todistuspuuta tulostuu.
Tulostetaan puu ASCII- muodossa.	Tulostuu
Tulostetaan puu latex- muodossa.	Tulostuu

Todistuksen rakentaminen:

Testitapaus	Odotettu tulos
Peruutetaan toiminto	Toiminto peruuntuu.
Klikataan navigointipuun solmua	Todistusikkuna kohdistuu solmun kohdalle.
Muutetaan fonttia.	Ok

7 Yhteenveto

Todistustoimittimen suunnitteleminen oliomallilla tyylikkäästi on osoittautunut varsin vaikeaksi, eikä saatu ratkaisu ole oliovinkkelistä katsottuna erityisen elegantti. Paremmin oliomainen ratkaisu olisi varmasti mahdollinen, mutta ei ole luultavaa että se olisi yhtään helpommin ymmärrettävä, paremmin todellisuutta vastaava tai laajennettava. Näyttää myös siltä että järjestelmän suunnitteleminen merkittävästi itsenäisemmillä komponenteilla pakottaa järjestämään useampia kirjanpitoja samasta rakenteesta, mikä ei voi olla laajennettavuuden, ymmärrettävyyden tai ylläpidettävyyden kannalta hyvä ratkaisu.

Suunnitteludokumenttia kirjoitettaessa huomasimme että lukujen määrän säilyttäminen parittomana saattaa olla todella vaikeaa. Juuri kyseisen ensisijaisen tärkeän suunnitteludokumentin piirteen säilyttäminen on syy tämän luvun olemassaoloon.

Hakemisto