Tampereen yliopisto

Sami-Santeri Svensk

XPATHIN SEMANTIIKKA GRAAFITIETOKANNOISSA CYPHER-KYSELYKIELELLE KÄÄNNETTYNÄ

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Maaliskuu 2021

TIIVISTELMÄ

Sami-Santeri Svensk: XPathin semantiikka graafitietokannoissa Cypher-kyselykielelle käännettynä
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Maaliskuu 2021

Erilaiset uudet sovellukset ja vaatimukset datan käsittelyyn ovat nostaneet useita perinteiselle relaatiotietokannalle vaihtoehtoisia tietokantajärjestelmiä keskusteluun ja käytäntöön viime vuosina. Yksi uusista viime vuosina yleistyneistä tietokantajärjestelmistä ovat graafitietokannat. Graafitietokannat ovat tulleet yhä suositummaksi tietokantajärjestelmäksi sosiaalisen verkostojen sivustojen, ja muiden graafimaiseen dataan pohjautuvien sovellusten myötä niin käytännössä kuin tietojenkäsittelytieteellisessä kirjallisuudessa.

Graafitietokantojen etuihin kuuluu korkean tason kyselykielten olemassaolo, vaikkakin graafikyselykielten kenttä on pirstaloitunut eikä standardikieltä ole. Tässä työssä ehdotetaan XPath-kielen soveltamista kyselyihin Neo4J-ominaisuusgraafitietokannassa. Perinteisesti yhden kyselykielen on ajateltu soveltuvan vain yhteen tietomalliin. XPath on hierarkkista tietomallia soveltaviin XML-dokumentteihin kehitetty kyselykieli. Vaikka XPath eroaa alkuperäisen sovellusalueensa ja syntaksinsa puolesta graafikyselykielistä, sillä tehdään samanlaisia kyselyitä XML-dokumenteissa kuin graafikyselykielet tekevät graafitietokannoissa. Työssä luodaan semanttiset säännöt XPathin kielen osille Cypher-kyselykielen kontekstissa attribuuttikieliopilla.

XPath mahdollistaa monipuolisen navigaation graafissa. Kyselyt XPathilla graafitietokannassa ovat syntaksiltaan loogisempia ja kompaktimpia kuin olemassa olevilla graafikyselykielillä. XPath on ilmaisuvoimainen kieli ja se pystyy ilmaisemaan kyselyitä graafiympäristössä kiitettävästi. Samaan aikaan XPath omaa hyvän suorituskyvyn. Lisäksi kieli on moderneja graafikyselykieliä tunnetumpi käytännössä ja tutkitumpi.

Avainsanat: XPath, NoSQL, Neo4j, Cypher, graafiteoria, graafitietokannat, attribuuttikielioppi

ABSTRACT

Sami-Santeri Svensk: Semantics of XPath in Graph Databases with Translation into Cypher Query Language
Master's thesis
Tampere University
Degree Programme of Computer Science
March 2021

A wide variety of new applications and new requirements for data have recently raised many NoSQL database systems into discussion and practice. Graph databases are among the trending NoSQL databases. Graph databases have gained popularity recently due to social networking services and other applications, where the data can be seen intuitively structured as graph. Graph databases have been examined recently in computer scientific literature. A huge advantage to graph databases is the existence of a high-level query language. Nevertheless, currently the field of modern graph query languages is fragmented. Traditionally, each query language is thought to be applicable to the specific data model. In this thesis, XPath, a well-known query language for tree-structured XML-documents, is suggested to query graph databases. XPath differs from modern graph query languages by its syntax and original data model. Still, it can express the same queries in tree-like data, such as graph pattern matching and navigating in tree-structured XML-documents, that modern graph query languages are used to query in the context of graph databases. In addition, XPath is more popular language than the new graph query languages. In this thesis, XPath is used to query Neo4J, the most popular graph database at the moment. Neo4J comes with high-level query language Cypher. The queries are translated from XPath into Cypher by using attribute grammar and hence creating semantics for XPath in the graph database. It will be shown that XPath is expressive language in the context of graph databases. XPath provides versatile properties for navigating in graph. Meanwhile queries can be expressed more intuitively, logically and in more compact form than in the existing graph query languages.

Keywords: XPath, NoSQL, Neo4j, Cypher, graph theory, graph databases, attribute grammar

1 Johdanto

Graafimaiseen dataan perustuvat sovellukset ovat tulleet yhä tärkeämmäksi osaksi ihmisten arkea ja herättäneet huomiota eri tieteenaloilla viime vuosina [Barceló et al., 2014; Libkin et al., 2016]. Graafeilla pystytään mallintamaan luonnollisesti mallinnettavan kohdealueen toimijat (entiteetit), ja niiden väliset suhteet toisiinsa. Graafeissa entiteetit mallinnetaan graafin solmuina ja näiden väliset suhteet kaarina [Angles, 2017]. Esimerkkejä graafimaiseen tietomalliin perustuvista sovelluksista ovat sosiaaliset verkostot [Almabdy, 2018], sekä biologian ja sosiaalitieteen sovellukset [Holzschuher and Peinl, 2016; Libkin and Vrgoč, 2012; Libkin et al., 2016; Kumar, 2015]. Graafimaista dataa luonnehditaan kompleksiseksi tai yhdistetyksi.

Graafimaisen datan tallentamiseen ja kyselemiseen voidaan käyttää [Barcelo, 2013], ja käytetäänkin [Angles, 2017], perinteistä relaatiotietokantaa. Relaatiomalliin [Codd, 1970] perustuvat relaatiotietokannat ovat olleet suosituin tietokantamalli jo 80-luvulta lähtien, ja säilyttäneet suosionsa tähän päivään asti [Frisendal, 2016; Almabdy, 2018]. Huolimatta nimestään, relaatiotietokannoissa relaatio painottaa attribuutin ja entiteetin välisen riippuvuuden kuvailua eikä kahden entiteetin välistä suhdetta [Frisendal, 2016]. Tästä syystä relaatiotietokannat ovat rajoittavia [Elmasri and Navathe, 2007] graafimaisen datan tallentamiseen [De Virgilio et al., 2013]. Relaatiotietokannan rajoittuvuuden takia graafimaisesta datasta tehtävät kyselyt ovat relaatiotietokannoissa käytettävällä SQL-kyselykielellä syntaktisesti monimutkaisia [Barcelo, 2013; Vainio ja Junkkari, 2014; De Virgilio et al., 2013]. Graafin sisältämien entiteettien välisiä suhteita hakevat kyselyt ovat oleellisia, kun kohdealue on esimerkiksi sosiaalinen verkosto. Sosiaalisessa verkostossa saatetaan haluta vaikkapa hakea henkilölle sellaiset kaverien kaverit, joilla on keskenään jo useita yhteisiä kaveria. Relaatiotietokannoissa SQL-kyselykielen semantiikka on epäselvä tällaisille kyselyille [Frisendal, 2016]. Tällaiset kyselyt ovat relaatiotietokannoissa myös laskennallisesti raskaita [Harrison, 2015]. Näiden ongelmien takia erilaisia laajennuksia on ehdotettu relaatiotietokantojen kyselykielelle [Vainio ja Junkkari, 2014].

Edellä kuvailtujen ongelmien ylitsepääsemiseksi tutkimus on viime vuosina kiinnostunut NoSQL-tietokannoista [Cure and Blin, 2015]. Termillä NoSQL viitataan sellaisten tietokantajärjestelmien joukkoon, jotka käyttävät relaatiotietomallista poikkeavaa tietomallia. Samalla ne pyrkivät erilaisin keinoin vastaamaan relaatiotietokantojen tunnettuihin puutteisiin [Tiwari, 2011; Cure and Blin, 2015]. NoSQL-tietokantojen katsotaan

jakautuvan neljään merkittävään tietokantatyyppiin: avain-arvo-, dokumentti-, wide column- ja graafitietokantoihin [Saake et al., 2018].

Erityisesti graafitietokannat ovat herättäneet huomiota [Angles, 2017]. Graafitietokannat ovat joukko tietokantoja, jotka tarjoavat tietomallinsa ansiosta vaihtoehdon relaatiotietokannoille graafimaisen datan tallentamiseen. Graafien tietomalli on intuitiivinen, ja se tarjoaa joustavan muodon graafimaisen datan mallintamiselle. Graafimaisesta mallista on helppo suorittaa tietynlaisia kyselyitä [Angles, 2017], kuten selvittää tietomallin rakenteen topologia [Angles and Gutierrez, 2008; Libkin et al., 2016]. Topologialla tarkoitetaan graafin rakennetta eli graafiin kuuluvien entiteettien suhteita toisiinsa.

Verrattuna muihin NoSQL-tietokantoihin, graafitietokannat näyttäytyvät Holzschuherin ja Peinlin [2016] mukaan erityisen kiinnostavan, sillä ne tarjoavat asianmukaisen kyselykielen. Graafikyselykieliä on tutkittu, ja niiden on todettu suoriutuvan hyvillä suoritusajoilla [Holzschuher ja Peinl, 2016; Almabdy, 2018]. Graafitietokantoja varten kehitetyt kyselykielet ovat herättäneet keskustelua suorituskykynsä [Holzschuher and Peinl, 2013] ja ilmaisuvoimansa osalta [Kostylev et al., 2018]. Vaikka varhaisimmat graafikyselykielet on esitelty jo 80-luvulla, ne ovat vielä suhteellisen tuore ilmiö [Angles and Gutierrez, 2008; Barceló, 2013]. Uutuutensa takia graafikyselykielet ovat tällä hetkellä marginaalissa niiden tunnettavuuden suhteen. Lisäksi iso osa graafitietokantojen kyselykielistä, pois lukien RDF-graafien [RDF, 2014] kyselykieli SparQL [SparQL, 2013], ovat standardoimattomia [Angles, 2018]. Tämän takia graafitietokannoille ei ole vielä olemassa sellaista vahvaa kieltä, kuin mitä SQL on relaatiotietokannoille, ja kehittäjäyhteisöllä on käytettävissä useita kieliä graafitietokantojen kyselemiseen. Viime vuosina on ollut ehdotuksia standardikielestä graafitietokannoille [Angles et al., 2018].

Tässä työssä ehdotetaan tunnetun ja perusteellisesti tutkitun [Gottlob et al., 2003; Benedikt and Koch, 2009; Gyssens et al., 2006] XPathin [XPath, 1999] käyttämistä kyselykielenä graafitietokannoissa. XPath on alun perin puumaisiin puolirakenteellisiin XML-dokumentteihin kehitetty kyselykieli. Työssä määritellään XPathin semantiikka eli kielen merkitys graafitietokannoissa. Semantiikan avulla tarkastellaan vastausta siihen, minkälaisia kyselyitä XPathilla voidaan tehdä, ja mitkä kielen osat ovat oleellisimpia graafitietokannoissa. Metodina tähän käytetään attribuuttikielioppia, jolla luodaan semanttiset säännöt XPath-kyselyille graafitietokannassa. Tässä työssä XPathia tullaan käyttämään tämän hetken suosituimman [DBEngines.com, 2021] graafitietokantahallintajärjestelmä Neo4j:n [Neo4J, 2020] kyselykielenä. Semantiikan määrittämiseksi XPath käännetään Neo4J-tietokannan tukemalle kyselykieli Cypherille [Cypher, 2020], koska

XPathilla ei voi ilman käännöstä tehdä kyselyitä Neo4J-graafitietokannasta. Vaikka XPath on kehitetty erilaisen tietomallin tarpeisiin, pystytään XPathilla silti ilmaisemaan samanlaisia operaatioita graafimaisesta datasta [Libkin et al., 2013] kuin Cypherillä ja muilla moderneilla graafikyselykielillä. XPathin syntaksia pidetään helposti opittavana ja kompaktina, koska kyselyt ovat lyhyitä ja helposti toistettavia. Lisäksi aiemmissa tutkimuksissa [Libkin et al., 2013; Libkin et al., 2016] XPathin kaltaisten kielten evaluoinnin aikavaatimus on todettu puuympäristössä lupaavaksi. Graafikyselykielenä XPathin kaltaisten kielten evaluointi on myös mahdollista suorittaa tehokkaasti saavuttaen silti hyvän ilmaisuvoiman [Libkin et al., 2013]. Tässä työssä verifioidaan Libkinin ja muiden [2013] ehdottama XPathin käyttö graafikyselykielenä.

Työssä osoitetaan myös, että samaa kyselykieltä voi käyttää kahdessa eri tietomallissa. Kyselykielten on perinteisesti ajateltu rajoittuvan tiettyyn tietomalliin [Mami et al., 2019]. Koska erilaiset tietomallit vastaavat erilaisiin tiedonsäilömistarpeisiin, on kyselykielten rajoittuneisuus yhteen tietomalliin käytännön kompastuskivi. Koska kyselykielet eroavat syntaksiltaan ja semantiikaltaan, voi uuden kielen opetteleminen olla haastavaa. [Mami et al., 2019]. Tietojenkäsittelytieteessä onkin alettu etsimään olemassa olevista kyselykielistä sellaisia kieliä, jotka tyydyttäisivät erilaisten tietomallien tarpeet mahdollisimman kattavasti [Mami et al., 2019].

Työn rakenne on seuraavanlainen. Luvuissa 2 ja 3 perehdytään työssä käsiteltyjen aiheiden teoriaan. Luvussa 2 tehdään katsaus graafitietokantoihin. Luvussa 3 taas tarkastellaan työn kannalta oleellisia kyselykieliä XPathia ja Cypheriä, sekä teoriaa niiden takana. Luvussa 4 esitetään XPathin semantiikan muodostaminen Cypher-kielelle. Semantiikan muodostamista varten määritetään luvussa 4 attribuuttikielioppi. Luvussa 5 esitetään valittujen XPath-ilmaisujen vastineet Cypher-kielen ilmaisuina. Esimerkit luvussa 5 myös havainnollistavat kuinka semantiikka muodostuu vaihe vaiheelta luvussa 4 määritetyllä attribuuttikieliopilla. Luvussa 6 esitetään kääntämistä varten luodun ohjelman toteutusperiaatteet. Luvussa 7 pohditaan XPathin semantiikkaa graafitietokannassa ja työhön valittuja ratkaisuja. Samalla luvussa 7 käsitellään työn aikana nousseita kysymyksiä, ja luodaan suuntaviivoja jatkokehitykselle.

2 Graafitietokannat

Tässä aliluvussa käsitellään graafitietokantoja. Graafitietokannat ovat NoSQL-tietokannoiksi luokiteltavia tietokantajärjestelmiä [Elmasri and Navathe, 2007]. Graafitietokannoille voidaan antaa formaali määritelmä graafiteorian graafirakenteen avulla [Angles,

2018]. Graafitietokannat eivät ole yksikäsitteinen joukko, eikä niille ole määritelty yhteistä tietomallia [De Virgilio et al., 2013]. Graafitietokannat nojaavat kuitenkin yhteisiin perusperiaatteisiin, joita on indeksivapaus ja tiedon tallentaminen ominaisuusgraafina [De Virgilio et al., 2013]. Graafitietokantojen teorialla on vahva pohja, sillä graafiteoria on matematiikan sivuhaara, jota on tutkittu ahkerasti [Harrison, 2015]. Graafitietokannoissa tapahtuvaan tiedonhakuun voidaan soveltaa perinteisiä graafiteorian graafioperaatioita [Angles and Gutierrez, 2008; van Bruggen, 2014].

Muiden NoSQL-tietokantojen tavoin graafitietokannat säilövät puolirakenteellista dataa [Elmasri and Navathe, 2007]. Puolirakenteellisesta datasta käytetään myös nimeä itsekuvaileva data, ja tarkoittaa dataa, joka sisältää myös kaavion [Buneman, 1997]. Graafitietokannoissa esimerkiksi instanssien suhteet ja niiden sisältämä data on instanssikohtaisia [Elmasri and Navathe, 2007]. Graafitietokannat yrittävät murtaa relaatiomallin rajoitukset ja mallintaa kohdealueiden luontainen graafirakenne graafina. Tällaisissa kohdealueissa datan yhdistyneisyys eli rakenteen topologia on yhtä tärkeää tai tärkeämpää kuin tietokannan sisältämä data [Angles and Gutierrez, 2008; Barceló, 2013]. Koska graafitietokannat perustuvat graafiteoriaan, se esitellään luvussa 2.1. Graafitietokannoissa tietomallina käytettävä ominaisuusgraafi esitetään aliluvussa 2.2

2.1 Graafiteoria

Rakenteena yksinkertainen graafi määritellään parina $G = (V, E)$. Määritelmässä V on graafiin kuuluvien solmujen joukko. E on solmuja yhdistävien kaarten joukko, niin että jokaisella kaarella pätee, että jos $(x, y) \in E$ niin $x \in V$ ja $y \in V$ [Koivisto ja Niemistö, 2018]. Tässä määritelmässä oletetaan, että jokainen graafiin kuuluva kaari yhdistää aina kaksi solmua. Kahden solmun välillä voi olla useita kaaria, jolloin tällaista graafia kutsutaan multigraafiksi. Kun kaari yhdistää solmuja x ja y, sanotaan, että solmut x ja y ovat toistensa vierussolmuja [Koivisto ja Niemistö, 2018]. Jos graafi on suunnattu, pari on järjestetty. Tällöin jokaisella kaarella on suunta parin ensimmäisestä solmusta x solmuun y. Kaaren voi ilmaista myös kolmen monikkona (v_0, e_0, v_1), missä $v_0, v_1 \in V$ ja e_0 on solmuja yhdistävä kaari. Jatkossa tarkastellaan suunnattuja graafeja.

Graafin solmuilla on asteluku, joka kuvaa solmusta lähtevien tai saapuvien kaarten lukumäärää. Solmun sisäaste kuvaa solmuun tulevien kaarten lukumäärää ja solmun ulkoaste solmusta lähtevien kaarten lukumäärää [Koivisto ja Niemistö, 2018]. Aste on siis yhtä suuri kuin solmun sisäasteen ja ulkoasteen summa.

Polku graafissa G on solmujen ja kaarien muodostama sekvenssi. Polku kulkee kahden graafiin kuuluvan solmun välillä, ja voidaan määritellä $v_0 \rightarrow v_n$. Määritelmässä v_0 on polun alkusolmu, v_n polun päätesolmu, ja n vastaa polun pituutta. Polun muodostava solmujen ja kaarten sekvenssi ilmaistaan $v_0e_0v_1e_1 \ldots e_{n-1}v_n$. Polkusana (tai pelkkä sana) on polkuun kuuluvien kaarten nimien joukko, missä sana kuuluu graafin määrittelemään aakkostoon Σ [Martens and Trautner, 2017]. Aakkosto Σ koostuu kaikkien graafin sisältämien kaarien nimistä. Kahden solmun välillä voi olla graafissa useita polkuja. Muutamia erityisiä polkuja ovat yksinkertainen polku, suora polku ja lyhin polku [Koivisto ja Niemistö, 2018]. Yksinkertainen polku on polku, jossa jokainen polkuun kuuluva kaari esiintyy ainoastaan kerran polussa. Suora polku tarkoittaa polkua, jossa kaikki polkuun kuuluvat solmut esiintyvät korkeintaan kerran polussa. Lyhin polku vaihtelee määritelmän mukaan. Yleensä lyhin polku on polku, jolla on pienin kaarten yhteenlaskettu paino. Vaihtoehtoisesti lyhimpänä polkuna voidaan pitää polkua, johon kuuluu vähiten kaaria. [Koivisto ja Niemistö, 2018].

Poluilla on graafitietokannassa tärkeä rooli. Graafitietokannan vahvimpana puolena pidetään kykyä kulkea graafissa eli navigoida. Navigointi tapahtuu graafin polkuja pitkin, jotka tarjoavat abstraktiotason kuvaamaan graafitietokannan entiteettien suhteita toisiinsa [Angles et al., 2018]. Graafitietokannassa navigointi on riippumaton graafitietokannan koosta ja tekee graafitietokannasta tietyissä skenaarioissa tehokkaita [De Virgilio et al., 2013].

Koska solmut ovat yhteydessä graafissa vain niiden vierussolmuihin, tämä yhteys ei sisällä tietoa siitä, kuinka muut graafin solmut kuin vierussolmut ovat saavutettavissa. Kahden solmun välinen polku löydetään käytännössä hakualgoritmeilla. Graafissa yleisimmin käytetyt hakualgoritmit ovat leveys- ja syvyyshaku [Koivisto ja Niemistö, 2018; Olsson and Magnusson, 2018].

Polkujen lisäksi graafitietokannoissa tärkeä osa on aligraafien löytämisellä. Graafin G aligraafi on sellainen graafin G osa, joka on graafin solmujen ja kaarten joukon osajoukko. Aligraafissa olevat solmut ja kaaret kuuluvat siis graafiin G. Formaalisti aligraafi määritellään parina $H = (W, F)$, missä $W \subseteq V$ ja $F \subseteq E$. [Koivisto ja Niemistö, 2018].

2.2 Ominaisuusgraafi

Ominaisuusgraafit (*property graph*) [Angles, 2018] ovat suunnattuja multigraafeja, jossa solmut kuvaavat entiteettejä ja kaaret entiteettien välisiä suhteita [Angles, 2018; van

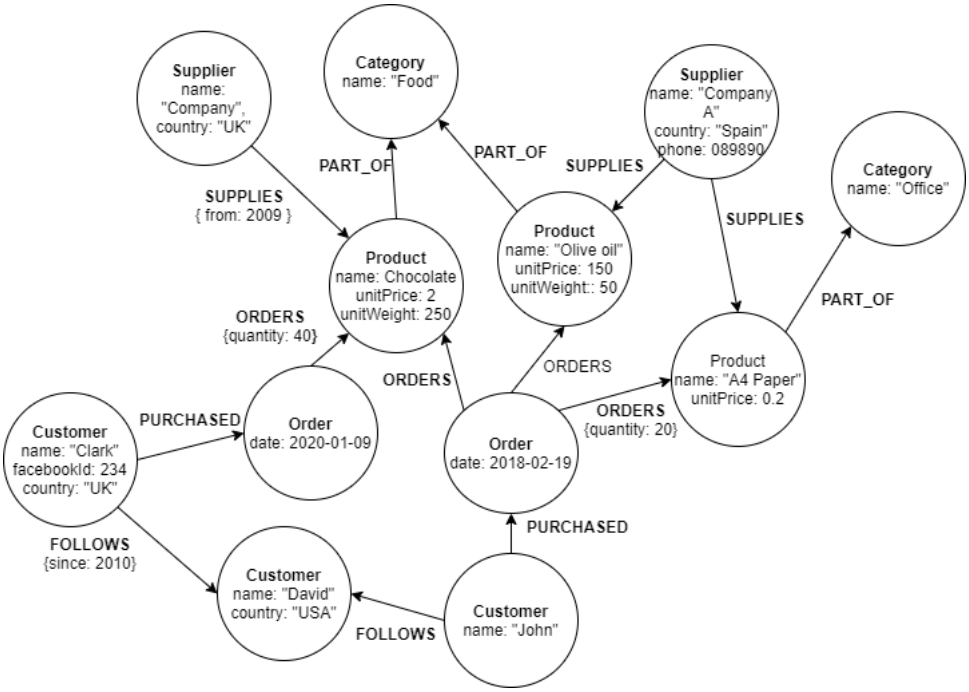
Bruggen, 2014]. Graafitietokantojen tietomallit perustuvat ominaisuusgraafeihin [De Virgilio et al., 2013]. Elmasri ja Navathe [2007] ovat luonnehtineet ominaisuusgraafitietomallin muistuttavan joiltain osin ER-mallia. Esimerkiksi Bordoloi ja muut [2014] aloittivat muuttamisen relaatiotietokannasta graafitietokannaksi ER-malli pohjana. Erona ER-malliin, graafitietokannoissa entiteettien sisältämät attribuutit ja niiden väliset suhteet muodostetaan ekstensionaalisella tasolla intensionaalisen tason sijaan.

Ominaisuusgraafia pidetään ilmaisuvoimaisena, koska siinä sekä graafiin kuuluviin solmuihin, että kaariin voidaan tallentaa dataa [Angles, 2018]. Ominaisuusgraafissa data tallennetaan joukkona avain-arvo-pareja [Elmasri and Navathe, 2007; Angles, 2018]. Täten ominaisuusgraafi on suoraviivainen abstraktio mallinnettavasta kohdealueesta [Angles et al., 2017]. Myös Angles ja Gutierrez [2008] ja van Bruggen [2014] näkevät ominaisuusgraafin luonnollisena tapana esittää graafimainen data. Angles [2018] on määritellyt ominaisuusgraafin formaalisti monikkona $G = (N, E, \rho, \lambda, \sigma)$.

1. N on tietokantaan kuuluvien solmujen joukko.
2. E on äärellinen joukko kaaria. Joukon N alkiot eivät voi kuulua joukkoon E.
3. $\rho: E \rightarrow (N \times N)$ on funktio, joka yhdistää kunkin graafiin kuuluvan kaaren kahden graafin solmujoukkoon N kuuluvan solmun välille.
4. $\lambda: (N \cup E) \rightarrow SET^{+} (L)$ on funktio, joka antaa solmulle tai kaarelle joukon nimikkeitä joukosta L.
5. $\sigma: (N \cup E) \times P \rightarrow SET^{+} (V)$ on funktio, joka antaa solmun tai kaaren ominaisuuteen P arvoksi osajoukon arvoja joukosta V.

Anglesin [2018] määritelmä tukee usean nimikkeen antamista solmuille ja kaarille. Lisäksi määritelmässä sallitaan usean arvon antamisen solmun tai kaaren ominaisuudelle. Määritelmä ei kuitenkaan ole ainoa määritelmä, sillä ominaisuusgraafeille ei ole standardimääritelmää [Angles et al., 2020]. Ominaisuusgraafia tietomallinaan käyttävät graafitietokannat saattavat käyttää eri määritelmää.

Kuvassa 1 on visuaalisesti kuvattu esimerkki ominaisuusgraafista. Kuvassa on nimetty määritelmän mukaan sekä solmut, että kaaret. Solmut ja kaaret voivat sisältää kokoelman ominaisuuksia. Ominaisuusgraafin puolirakenteellinen tietomalli ilmenee kuvassa 1 siten, että saman entiteettityypin entiteetit tai saman suhdetyypin suhteet saattavat sisältää keskenään erilaisen kokoelman ominaisuuksia.



Kuva 1. Esimerkki ominaisuusgraafista

Ominaisuusgraafin voi katsoa pohjautuvan RDF-graafiin, sillä ominaisuusgraafissa entiteettiin kiinnitettyä ominaisuuksien listaa voi luonnehtia samanlaiseksi kuin RDF-graafin solmun predikaatteja. Attribuutin arvoa ominaisuusgraafissa voi pitää samana kuin RDF-graafin objektia. Näillä kahdella graafimallilla on kuitenkin eronsa [Angles et al., 2020]. Ominaisuusgraafi tietomallina soveltuu parhaiten sellaiselle rakenteelle, joka on luonnostaan suunnattu multigraafi, ja jossa sekä solmut, että kaaret varastoivat dataa [van Bruggen, 2014].

Ominaisuusgraafia käyttävistä tietokantahallintajärjestelmistä suosituin on Neo4J [DBEngines.com, 2021]. Neo4J:n erityisehtoina on, että solmulla voi olla mielivaltainen määrä nimikkeitä, mutta suhteessa nimikkeitä voi olla täsmälleen yksi. Neo4J:ssä jokainen solmu ja kaari on indeksoitu. NoSQL-tietokantojen tapaan Neo4J ei vaadi käytettäväksi kaaviota, mutta tarjoaa mahdollisuuden luoda sellainen [Neo4J, 2020].

3 Työssä tarkasteltavat kyselykielet

Tässä luvussa käsitellään tässä työssä tarkasteltavia kyselykieliä. Kyselykielet ovat kokoelma operaattoreita tai päättelysääntöjä, joita voidaan soveltaa tietomallin tietorakenteessa. Kyselykielten tavoitteena on manipuloida ja kysellä tietoa missä tahansa rakenteen yhdistelmissä [Angles and Gutierrez, 2008]. Libkin ja muut [2016] sanovat, että jokaisen tietomallin perustehtävä on se, miten tietomallin sisältämää dataa ja suhteita voidaan ky-

sellä. Holzschuher ja Peinl [2016] näkevät korkean tason kyselykielen olemassaolon olleen avaimena tietokantajärjestelmien suosioon. Holzschuher ja Peinl [2016] mainitsevatkin, että kehittynyt kyselykieli SQL ja sen laajennukset ovat auttaneet relaatiotietokantoja vakiintumaan parhaaksi vaihtoehdoksi datan tallentamiseen.

Kyselykielen on tärkeää säilyttää tasapaino tehokkuuden ja ilmaisuvoiman välillä. Ilmaisuvoima luonnehtii tarkkuutta, millä käsitteet ja niiden väliset suhteet voidaan määritellä [Cure and Blin, 2015]. Kielen täytyy tukea kyselyitä, jotka ovat tärkeitä tietomallin rakenteen kannalta [Libkin et al., 2016]. Samalla kyselykielen tulee olla tehokas, jolloin evaluoinnin aikavaatimuksen tulee olla matalaa [Libkin et al., 2016]. Evaluointi tarkoittaa vastauksen tuottamista kysymykseen, ja se on tietomallien perustehtävä [Kostylev et al., 2016]. Formaalisti evaluointi voidaan määritellä kysymyksenä, onko $h \in q(G)$ [Angles et al., 2017]. Määritelmässä on annettu kysely q, mahdollinen vastaus h ja tietokanta G, ja tarkoituksena on ratkaista, onko vastaus h kyselyn Q vastaus tietokannassa G. Kyselykielten yleisiä ominaisuuksia ovat projektio, suodatin ja yhdiste. Projektio määrittää kyselystä palautettavan muuttujan. Suodattimella asetetaan lisäehtoja, joilla karsitaan tuloksia kesken kyselyn käsittelyn. Yhdiste on joukko-opillinen operaatio, joka palauttaa usean alikyselyn tulokset samassa tulosjoukossa.

Tämän luvun aliluvuissa käsitellään kyselykielten teoriaa, ja työn kannalta oleellisia kyselykieliä XPathia ja Cypheriä. Aliluvussa 3.1 esitellään säännölliset polkukyselyt, johon useimpien graafikyselykielten teoria perustuu. Aliluvussa 3.2 esitellään XPath. Aliluvussa 3.3 esitellään graafikyselykielet, niiden teoria ja graafikyselykieli Cypher.

3.1 Säännölliset polkukyselyt (RPQ)

Säännöllinen polkukysely (englanninkielisessä kirjallisuudessa *regular path query*, ja tästä eteenpäin RPQ) on kyselymekanismi, johon navigointi moderneissa graafikyselykielissä perustuu [Calvanese et al., 2003; Kostylev et al., 2018; Angles et al., 2017]. RPQ mahdollistaa graafin kyselemisen mielivaltaisen pitkiä poluilla [Martens and Trautner, 2017]. RPQ-kyselyitä ei ole tyypitetty [Colazzo and Sartiani, 2015]. RPQ-mekanismilla haetaan graafitietokannasta solmupareja, joita yhdistää suunnattu polku. Formaali määritelmä RPQ:lle on säännöllinen lauseke L tai monikkona ilmaistuna P = (x, L, y). Määritelmässä P kuvaa polkua, x on polun alkusolmu, y on polun loppusolmu ja L on säännöllinen lauseke. Säännöllinen lauseke L kuuluu tietokannan määrittämään aakkostoon Σ [Barcelo, 2014].

RPQ:t ovat pohjimmiltaan säännöllisiä lausekkeita [Bagan et al., 2012; Martens and Trautner, 2017]. Säännöllinen lauseke on merkkijono, joka joko vastaa tai on vastaamatta jotain toista merkkijonoa. Esimerkiksi säännöllinen lauseke *abc* vastaa merkkijonoja *abc* ja *aabcd*, koska säännöllisen lausekkeen sisältämät merkit kuuluvat muihin merkkijonoihin lausekkeessa annetussa järjestyksessä. Säännöllinen lauseke *abc* ei kuitenkaan esimerkiksi vastaa merkkijonoa *bca*, koska merkkien järjestys on väärä. Säännöllisiin lausekkeiden ilmaisuvoimaa voi parantaa erilaisilla operaattoreilla. Esimerkiksi säännöllinen lauseke a^+ täsmää merkkijonoihin, jotka sisältävät ainoastaan a-kirjaimia riippumatta merkkijonon pituudesta. Osa näistä operaattoreista esitetään taulukossa 1 RPQ:n kontekstissa. Säännöllisen lausekkeen tapaan RPQ:ta verrataan graafitietokannan polkuihin ja niiden muodostamiin sanoihin. Vaikka RPQ perustuu säännöllisiin lausekkeisiin, RPQ:lla ja säännöllisillä lausekkeilla on semanttiset eronsa, joka tulee esille, jos ketjutetaan useita RPQ-kyselyitä konjunktiolla. Konjunktiot eivät kasvata säännöllisten lausekkeiden ilmaisuvoimaa, mutta konjunktio lisää RPQ-kyselyiden ilmaisuvoimaa graafeissa, koska graafissa kahden solmun välillä voi kulkea useampi polku [Vardi, 2016]. Näin ollen konjunktiolla voidaan hakea joko kaikki ehdot täyttävä yksi polku, tai useampi polku, niin että kukin polku täyttää aina yhden konjunktion ehdon.

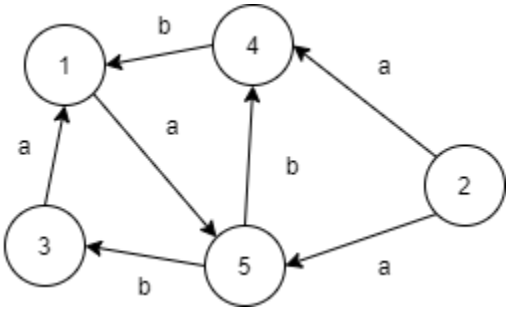
Graafitietokannoissa RPQ-mekanismia käytetään vastaamaan kysymykseen, onko graafitietokannassa olemassa RPQ:n ilmaisemia nimettyjä kaaria vastaavaa polkua. Polkuun määritetään myös polun alku- ja loppusolmu tutkimaan, onko loppusolmu saavutettavissa alkusolmusta käsin. Polkujen olemassaolon testaus kuuluu graafitietokantojen oleellisimpiin tehtäviin [Angles et al., 2017]. Henglein ja Nielsen [2011] muistuttavat, että säännöllisten lausekkeiden yhteydessä ei yleensä haluta saada vain kyllä/ei-vastausta täsmäyksestä, vaan asianmukaiset osat, jotka täsmäävät säännöllisen lausekkeen ilmaisuun. Samoin RPQ:ta käytetään palauttamaan polun olemassaolon selvittämisen yhteydessä ne graafin osat, jotka täsmäävät RPQ-ilmaisun kanssa. RPQ-mekanismilla palautetaan graafista kaikki solmuparit, jotka mukautuvat RPQ-ilmaisuun [Wood, 2012]. Calvanese ja muut [2000] täsmentävät näiden palautettavien solmuparien olevan sellaisia, joiden välinen polkusana täsmää RPQ-ilmaisuun.

Kuten aiemmin todettiin, säännöllisten ilmaisujen operaattorit soveltuvat RPQ-ilmaisuihin, ja niillä voidaan monipuolistaa graafissa navigointia. Näiden operaattorien mukaiset ilmaisut on kuvattu taulukossa 1.

RPQ-ilmaisu	Kuvaus
a	hakee graafista kaikki polut, jotka sisältävät vain ja ainoastaan yhden kaaren, ja jonka nimi on a
a*	hakee kaikki polut, jotka sisältävät mielivaltaisen pituisen sekvenssin, joka koostuu a-nimisistä kaarista. Lisäksi sekvenssi voi olla tyhjä, jolloin polku on tyhjä polku lähtösolmuun itseensä.
a+	hakee kaikki polut, joihin kuuluvien kaarten nimet ovat kaikki a-nimisiä, ja missä a-niminen kaari esiintyy ainakin kerran eli polku ei voi olla tyhjä
a·b	hakee kaikki polut, jossa tulee ensin esiintyä kerran kaari nimeltään a, ja sitä seurata kaari nimeltään b.
a{i,j}	Hakee kaikki polut, joissa alkusolmusta voi kulkea peräkkäin a-nimistä kaarta vähintään *i* kertaa eteenpäin, mutta korkeintaan *j* kertaa peräkkäin a-nimistä kaarta
*****	Hakee kaikki polut, jonka säännöllinen ilmaisu voi olla minkä tahansa graafin aakkostoon kuuluvan merkkijonon mukainen.

Taulukko 1. RPQ-ilmaisut

Kuvassa 2 esitetään graafi, jossa kaaret on nimetty kirjaimilla graafin määrittävästä aakkostosta Σ = (a, b). Kuvan 3 solmut on yksilöity tunnistamisen helpottamiseksi kokonaisluvuilla.



Kuva 2. Graafi, jossa solmut on yksilöity numeroin, ja kaarilla on nimi

Kuvassa 2 kyselyn a palauttama solmuparien joukko olisi {(3,1), (1,5), (2,5), (2,4)}. Kysely a^* palauttaisi joukon {(1,1), (1,5), (2,2), (2,4), (2,5), (3,3), (3,1), (3,5)}. a^+ palauttaisi saman joukon kuin edeltävä kysely, mutta ilman solmupareja, joissa parin molemmat solmut ovat samat. ab palauttaisi {(1,4), (1,3), (2,1), (2,3), (2,4)}. Kysely $a^{\{3,5\}}$ palauttaa tyhjän joukon solmupareja, koska yksikään polku graafissa ei sisällä peräkkäin kolmea a-nimistä kaarta.

RPQ-operaattoreista * ja + sisältävät transitiivisen sulkeuman. Tämä ominaisuus auttaa määrittelemään rekursiivisesti mallin ominaisuuksia ja määrittelemään saavutettavuutta [Bergmann et al., 2012]. Transitiivisuus tarkoittaa, että joukossa D on olemassa alkioiden pari (u, v), ja sekvenssi $p_0p_1p_2 \ldots p_{n-1}p_n$ missä $n \in N$, $u = p_0$ ja $v = p_n$, ja kaikki parit $p_{a-1}p_a$ kuuluvat relaatioon R [Bergmann et al., 2012]. Transitiivinen sulkeuma relaa-

tiolle R on kaikki joukon parit, joiden välillä on transitiivinen suhde. Esimerkki transitiivisesta sulkeumasta voisi olla lentomatkat, joka muodostuu relaation lentoyhteys transitiivisista suhteista. Esimerkissä relaatio kuvaa lentokenttiä, joiden välillä on suora lentoyhteys. Transitiivinen sulkeuma lentomatka kuvaa tällöin kaikkia yhteyksiä, jotka ovat mahdollisia suoraan lentäen tai tekemällä välilaskun lentokentällä.

RPQ ei luonnostaan pysty tukemaan useita luonnollisilta tuntuvia polkuoperaatioita, joten RPQ:lle on olemassa ilmaisuvoimaa lisääviä laajennuksia [Angles et al., 2017]. Yksi RPQ:n laajennus on kaksisuuntainen RPQ, joka sisältää käänteiseen suuntaan kulkemisen mahdollistavan operaattorin $^-$ [Libkin et al., 2013; Kostylev et al., 2018]. Esimerkiksi operaatio a^- hakee polut, joilla lähtösolmusta maalisolmuun kuljetaan yhtä a-nimistä kaarta taaksepäin. Näin ollen kaksisuuntaiseen RPQ-ilmaisun sisältävän kyselyn vastaukseksi evaluoidaan solmuparien joukko, jotka ovat suhteessa toisiinsa kaarisekvenssillä kaaria eteen- tai taaksepäin kulkien. Koska RPQ perustuu graafin topologiaan, RPQ ei pysty ilmaisemaan kyselyitä, jotka yhdistävät dataa ja topologiaa [Libkin and Vrgoč, 2012]. Ominaisuusgraafi kuitenkin sisältää solmujen ja kaarien lisäksi dataa, jonka takia polulle voi olla mielekästä asettaa polkusanan lisäksi muita rajoituksia [Libkin et al., 2016]. Vaikka RPQ ei itsessään huomioi dataa, moderneissa graafikyselykielissä on keinot datan esittämiseen kyselyissä.

3.2 XPath

Tässä aliluvussa esitellään XPath. XPath on tiedon hakuun XML-dokumenteista kehitetty kyselykieli [XPath, 1999; Melton and Buxton, 2006]. XPathista puhuttaessa viitataan W3C-konsortion standardoimaan kieleen, joka kehittyy jatkuvasti. XPathista on julkaistu useita versioita ensimmäisen version jälkeen, ja viimeisin XPathin versio 3.1 on vuodelta 2015. Ensimmäinen XPathin versio 1.0 julkaistiin 1999 [XPath, 1999]. Versio 1.0 on melko laaja kieli sisältäen kattavan kokoelman navigoinnillisia ja matemaattisia operaatioita. XPath näyttelee isoa roolia myös muissa XML-teknologioissa, kuten XSLT, XPointer ja XQuery [Bárcenas et al., 2015; Gottlob et al., 2003]. Version 1.0 jälkeisissä versioissa kieli on muuttunut huomattavasti, joka laajentaa kielen toimintoja huomattavasti ja vie XPathia enemmän ohjelmointikielen suuntaan. Kuitenkin iso osa tutkimuksesta keskittyy edelleen nimenomaan XPathin versioon 1.0 [Benedikt and Koch, 2009].

Sekä Benedikt ja Koch [2009], että Libkin ja muut [2013] nostavat esiin XPathin syntaksin läheisyyden loogisiin kieliin, kuten predikaattilogiikkaan. Myös Bárcenas ja

muut [2015] mainitsevat XPathin olevan ilmastavissa kaksipaikkaisella predikaattilogiikalla. Useiden kyselykielten ilmaisut voidaan ilmaista predikaattilogiikan ilmaisuina, jonka takia predikaattilogiikkaa käytetään yleisesti mittatikkukielenä kyselykielille niiden ilmaisuvoiman mittaamiseen [Libkin et al., 2013]. Näin ollen XPathia voi pitää ilmaisuvoimaisena kielenä. Ilmaisuvoiman lisäksi evaluointi onnistuu XML-dokumenteissa laskennallisesti tehokkaasti [Libkin et al., 2013]. Benedikt ja Koch [2009] ovatkin arvelleet XPathin olevan yhtenä syynä myös XML:n suosioon.

Aliluvussa 3.2.1 esitellään XPathin alkuperäinen kohdealue XML. Aliluvussa 3.2.2 perehdytään XPathin tietomalliin ja evaluointiin. Aliluvussa 3.2.3 perehdytään kielen navigoinnillisen ytimen muodostaviin akseleihin. Aliluvussa 3.2.4 esitellään esimerkkikysely XPathilla.

3.2.1 XML

XML (*eXtensive Markup Language*) on merkkauskieli, jolla voidaan kuvailla mallinnettavaa kohdealuetta hierarkiaan perustuvana dokumenttina [XML, 2008]. XPath lukeutuu itsekuvaileviin tietomalleihin.

XML-dokumentti koostuu elementeistä, joiden tunnisteena on merkkijono [XML, 2008]. XPathin elementit sisältävät merkkidataa tai merkintäkieltä. Elementtejä on XML:ssä kahdenlaisia: kompleksiset elementit ja yksinkertaiset elementit [Elmasri and Navathe, 2007]. Kompleksiset elementit rakentuvat hierarkkisesti muista elementeistä. Yksinkertaiset elementit koostuvat literaaleista [Elmasri and Navathe, 2007]. Näin ollen XML-dokumentin voi nähdä puurakenteena, jossa kompleksiset elementit ovat puun sisäsolmuja ja yksinkertaiset elementit puun lehtisolmuja [Elmasri and Navathe, 2007]. XML:ssä hierarkkista suhdetta elementtien kesken kutsutaan lapsi-vanhempi-suhteeksi. Elementin sisältävään elementtiin viitataan XML:ssä vanhempana ja sisällössä olevaan elementtiin lapsena [XML, 2008]. Kaikilla XML-dokumentin elementillä, paitsi juurielementillä, on yksi vanhempi. Kaikilla dokumentin elementeillä voi olla useita lapsielementtejä. Hierarkkisen rakenteen turvin XML:ssä voidaan määritellä elementtien välille transitiivisia suhteita esivanhempi ja jälkeläinen.

Puumainen malli on ilmaisuvoimainen itsekuvailevan datan tarpeisiin [Grust et al., 2004]. XML:n voi nähdä myös suunnattuna graafina [Elmasri and Navathe, 2007]. Tällöin elementit nähdään solmuina, joista lähtee nimeämätön kaari niiden lapsisolmui-

hin. XML-dokumentin voi käsittää silmukattomana yhdistettynä graafina, joka ei ole multigraafi eikä siinä ole olemassa kaaria, joilla on sama lähtö- ja tulosolmu [Pettovello and Fotouhi, 2006].

Rakenteen lisäksi XML-elementit voivat sisältää joukon attribuutteja. Attribuuteilla annetaan lisätietoa elementistä literaaleina [Elmasri and Navathe, 2007]. Attribuutit ovat avain-arvo-pareja. [XML, 2008].

Kuvassa 3 esitetään esimerkki XML-dokumentista. Kuvan 3 esimerkki kuvaa kohdealuetta, johon kuuluu henkilöitä, asuinmaa ja tieto henkilön omistuksista.

```
<henkilö>
	<omistaa>
		<auto />
	</omistaa>
	<asuu>
		<maa @nimi=”Suomi”></maa>
	</asuu>
</henkilö>
<henkilö>
	<asuu>
		<maa @nimi=”Suomi”></maa>
	</asuu>
</henkilö>
```

Kuva 3. XML-dokumentti

3.2.2 Tietomalli ja evaluointi

XPath toimii XML-dokumentin abstraktilla loogisella rakenteella sen käyttäjälle näkyvän syntaksin sijaan [XPath, 1999; Holzner, 2004]. XPath pelkistää XML-dokumentin solmuista koostuvaksi puurakenteeksi [XPath, 1999; Holzner, 2004]. Erityyppisiä solmuja XPathin tietomallissa ovat juurisolmut, elementtisolmut, attribuuttisolmut, tekstisolmut, kommenttisolmut, nimiavaruussolmut ja prosessiohjesolmut [XPath, 1999; Gottlob et al., 2003]. XPathin solmutyypit vastaavat XML:ssä esiintyviä ominaisuuksia.

XPath-ilmaisu on kuvaus alkusolmusta loppusolmuun. XPathissa valitaan kyselyn ehdot täyttävät XML-dokumentin elementit [Melton and Buxton, 2006]. XPath-ilmaisu palauttaa arvon, joka on tietotyypiltään joko solmujoukko, merkkijono, numeerinen arvo tai totuusarvo [XPath, 1999; Gottlob et al., 2003]. Ilmaisu koostuu poluista, ja polut koostuvat askelista. Askel koostuu solmusta, jotka ovat XPathin tietomallin solmuja. Askeleita erotetaan /- tai //-merkeillä. Askelia erottavia merkkejä käytetään myös ilmaisun alussa. / tarkoittaa ilmaisun aloittamista rakenteen juurisolmusta. //-notaatio alustaa kontekstisolmuksi minkä tahansa askelen ehtoja täsmäävän solmujoukon. Askel sisältää kolme osaa: akseli, solmutesti ja predikaatti, joiden järjestyksen on oltava kuvailtu järjestys. Askeleen

pakolliset osat ovat akseli ja solmutesti. Akseli määrittää navigaatiosuunnan edellisestä kontekstisolmusta. Solmutesti määrittää solmun nimen, johon edellisestä kontekstisolmusta siirrytään. Solmutestinä voi olla merkkijono tai villi kortti, jolloin solmutesti hyväksyy minkä nimisen elementin tahansa [Gottlob et al., 2003]. Predikaatti on askelen valinnainen osa, ja kirjoitetaan XPathin syntaksissa hakasulkeiden sisälle. Predikaatilla voidaan antaa solmulle ylimääräinen ehto tai useampia ehtoja solmutestin lisäksi. Predikaatti on pohjimmiltaan totuusarvon palauttava funktio, joka saa syötteenä XPath-ilmaisun ja evaluoi sen totuusarvoksi. Predikaatin palauttaessa totuusarvon epätosi, predikaattia edeltänyt ilmaisu epäonnistuu eikä koko XPath-kysely täten palauta mitään. Kun predikaatti palauttaa arvon tosi, se suodattaa pois tulokset, jotka eivät täytä predikaatin sisältämää ehtoa. Predikaatti voi sisältää yhden ehdon tai useita ehtoja loogisilla operaattorilla ketjutettuna. Koska predikaatti sisältää XPath-ilmaisun, predikaatti voi sisältää sisäkkäisiä predikaatteja. Täten XPath soveltuu ilmaisemaan hyvinkin kompleksisia polkurakenteita ja polkujen haarautumista. Predikaatti voi esiintyä myös hakasulkeiden ulkopuolella polun päätteenä, jolloin koko XPath-ilmaisu palauttaa totuusarvon. XPathin syntaksi noudattaa monia tuttuja käytäntöjä esimerkiksi URI-syntaksista ja tiedostopoluista [Cassidy, 2003; XPath, 1999].

XPath-ilmaisua prosessoidaan kontekstin avulla. Kontekstilla tarkoitetaan ilmaisun osaa, joka on prosessoitavana. Tätä ilmaisun osaa kutsutaan kontekstisolmuksi, ja on aina yksi XPath-ilmaisun sisältämä askel. Kontekstisolmu tuottaa joukon solmuja, jotka vastaavat askelen solmutestin ja predikaatin ehtoihin. Samalla joukon solmut ovat askelen akselin kuvailemalla navigaatiolla saavutettavissa edellisestä kontekstisolmusta. Kontekstisolmu vaihtuu aina, kun siirrytään askeleesta seuraavaan. Myös predikaattiin siirryttäessä kontekstisolmu siirtyy samalla tavalla kuten askelen yhteydessä. Predikaatista poistuttaessa kontekstisolmuksi asetetaan väliaikaisesti predikaattia edeltänyt kontekstisolmu. Ilmaisun prosessointi on päättynyt, kun kontekstisolmuksi ei voi sijoittaa uutta solmua.

XPath-ilmaisun rakenne vaikuttaa kyselyn evaluointiin. XPath-ilmaisusta palautettava arvo on kyselyssä oikeimman puoleisin elementti, joka on predikaatin ulkopuolella. Erillistä projektiolauseketta ei ole määritetty XPathissa, vaan ilmaisun projektio tulkitaan ilmaisussa implisiittisesti. Elmasri ja Navathe [2007] pitävät tätä XPathin heikkoutena. Samalla kyselyn luettavuus heikkenee. Toisaalta XPath-ilmaisut säilyvät lyhyinä ja kompakteina. XPath sallii tiedon aggregoinnin funktioiden avulla. Funktion ollessa läsnä,

parametrina saadun XPath-ilmaisun palauttama arvo aggregoidaan funktion mukaan. Lisäksi XPathissa voi käyttää kaksipaikkaisia merkkijonofunktioita, joita voi käyttää tulosten suodattamiseen predikaateissa merkkijonojen ominaisuuksien perusteella [XPath, 1999].

3.2.3 Akselit

XPathin navigaatio perustuu akseleihin. Bárcenas ja muut [2015] katsovat akselien vastaavan RPQ-operaattoreita. Kaikki XPathin käyttämät akselit ja niiden kuvaukset on lueteltu taulukossa 2.

Akseli	Kuvaus
Ancestor	Hakee kontekstisolmun kaikki elementit, jotka ovat joko kontekstisolmun vanhempia tai esivanhempia.
Ancestor-or-self	Ancestor-or-self lisää ancestor-akseliin refleksiivisyyden. Tällöin kontekstin solmujoukkoon kuuluu sekä kontekstisolmu, että vanhempi- ja esivanhemmat.
Attribute	Indikoi kontekstisolmuun liittyviä attribuutteja. Attribuutit ovat avain-arvo-pareja, joiden arvo voi olla merkkijono tai numero. Attribuutin voi merkitä XPathissa lyhennetyssä muodossa @-notaatiolla. Attribuutti on polun päätepiste, eikä lukeudu kuuluvaksi kontekstin solmujoukkoon. Kuitenkin attribuuttiin voi aina lisätä operaattorin seuraamaan attribuutin nimeä.
Child	Hakee kontekstisolmulle lapsisolmut. Lapsisolmu on kontekstisolmua hierarkiassa välittömästi seuraavalla tasolla oleva solmu. Jos XPath-ilmaisussa akselia ei ole mainittu askeleessa, child-akseli on oletus askeleen akseliksi.
Descendant	Hakee kontekstisolmusta katsottuna kaikki elementit, jotka ovat sen jälkeläisiä, lapsisolmut mukaan luettuna. Lyhennetyssä muodossa akselin voi ilmaista //-notaatiolla, ja hakee solmut miltä tahansa rakenteen syvyydeltä.
Descendant-or-self	Descendant-or-self lisää descendant-aksellin refleksiivisyyden ja palauttaa kaikki kontekstisolmun jälkeläiset sekä kontekstisolmun.
Following	Valitsee kaikki elementit dokumentista, jotka esiintyvät kontekstisolmun lopputunnisteen jälkeen.
Following-sibling	Valitsee solmut, joilla on kontekstisolmun kanssa sama vanhempisolmu, ja ovat sijoittuneet järjestyksessä kontekstisolmun jälkeen.
Namespace	Palauttaa solmut, jotka ovat nimiavaruussolmuja.
Parent	Palauttaa kontekstisolmun vanhemman. Vanhempielementti on suoraan kontekstisolmua hierarkiassa ylempi elementti. Koska XML on hierarkkinen rakenne, kullakin elementillä voi olla vain yksi vanhempi. Lyhennetyssä muodossa parent-akseli on ilmaistavissa kahdella pisteellä, jolloin akselin yhteydessä olevalle solmutestille ei aseteta ehtoja.
Preceding	Valitsee kaikki elementit dokumentista, jotka ennen kontekstisolmun alkutunnistetta.

Preceding-sibling	Valitsee kaikki solmut, joilla on sama vanhempi kuin kontekstisolmulla, mutta ovat sijainniltaan ennen kontekstisolmua.
Self	XPathissa self-akselilla kuvataan kontekstisolmua. Self-akselin voi ilmaista lyhennettynä pistenotaatiolla ”.” .

Taulukko 2. XPathin akselit

3.2.4 Esimerkkikysely

Esimerkkikysely XPathilla esitetään kyselyssä 1. Kysely 1 hakee henkilöt, jotka omistavat auton, ja henkilö asuu maassa nimeltään Suomi. Kyselyssä 1 ilmaisu sisältää XPathin ominaisuuksista predikaatin, solmutestin, askelet sekä attribuuttien arvon testauksen.

```
//henkilö[omistaa/auto and asuu/maa/@nimi=”Suomi”]
```

Kysely 1. Esimerkki XPath-ilmaisusta

Kysely 1 alkaa `//`−notaatiolla, joka alustaa kaikki rakenteesta löytyvät henkilönimiset elementit kontekstisolmuksi. Toisin sanoen kontekstisolmun solmujoukkoon sijoitetaan siis kaikki henkilö-elementit. Koska XPathissa predikaatit yhdistetään polkuun ilman erillistä lauseketta, voidaan predikaatteja antaa sitä mukaa kuin kysely etenee, kuten kyselyssä 1 on tehty, jossa lukusuunnassa vasemmalta oikealle esiintyy ensin henkilö-elementteihin liittyvä ehto, jonka jälkeen ilmaisuun kuuluu auto-solmu. Auto-solmu kuvaa kaikkia auto-elementtejä. Operaattori and yhdistää usean ehdon kuuluvaksi samaan predikaattia edeltäneeseen solmuun. Kysely 1 on esimerkki konjunktiivisesta kyselystä, jossa asetetaan useita väittämiä, joiden jokaisen täytyy pitää paikkansa, jotta predikaatti palauttaa totuusarvon tosi, ja että kysely onnistuu. Esimerkissä henkilön täytyy omistaa auto, ja henkilön täytyy myös asua maassa nimeltään Suomi. Yhdenkin näiden väittämien epäonnistuminen johtaa koko kyselyn epäonnistumiseen.

Kun kysely 1 tehdään kuvan 3 mukaiseen XML-dokumenttiin, palautetaan dokumentin järjestyksessä ensimmäinen Henkilö-elementti. Dokumentista voi havaita, että elementin kuvaama henkilö omistaa auton, ja maan nimi, jossa henkilö asuu, on Suomi. Kuvan 3 järjestyksessä toisen Henkilö-elementin kuvaama henkilö asuu myös Suomessa. Mutta koska hän ei omista autoa, konjunktion molemmat puolet eivät ole tosia, jolloin elementtiä ei palauteta.

3.3 Graafikyselykielet

Tässä aliluvussa käsitellään graafikyselykieliä, jotka ovat graafitietokantoja varten kehitettyjä kyselykieliä. Tämän hetken suosituimmat graafikyselykielet ovat Cypher, SparQL ja Gremlin, joista suosituin graafikyselykieli on Neo4j-tietokantaan tiiviisti sidoksissa

oleva Cypher [Angles et al., 2017]. Aliluvussa 3.3.1 käsitellään graafikyselykielten teoria. Aliluvussa 3.3.2 esitellään Cypher-kyselykieli.

3.3.1 Graafikyselykielten teoria

Graafikyselykieliä on useita. Vaikka graafikyselykielet eroavatkin toisistaan ominaisuuksiltaan, kuten syntaksiltaan ja ilmaisuvoimaltaan, niillä pitävät sisällään yhteisen kahteen operaatioon perustuvan käsitteellisen ytimen. Nämä operaatiot ovat graafihahmon täsmäys ja navigoiminen graafissa [Angles et al., 2017].

Graafihahmolla määritellään rakenteelliset vaatimukset, jonka graafin sisältämän aligraafin täytyy tyydyttää tuottaakseen onnistuneen täsmäyksen [Gallagher, 2006]. Formaalisti täsmäys määritellään graafihahmon Q = (W, F) kartoituksena m graafissa G = (V, E). Täsmäyksessä kartoitetaan vakioiden (C) ja muuttujien (A) yhdiste $C \cup A$ vakioiden joukkoon, niin että kaikille $a \in C$ kartoituksella m(a) = a kartoitetaan vakiot itseensä. Kartoituksella tarkoitetaan vastaavuuksien etsimistä kahden eri joukon elementtien välillä. Muuttujilla tarkoitetaan kaikkia käyttäjän määrittelemiä graafitietokannan osia kyselyssä. Tällaisia osia voi esimerkiksi olla entiteetin nimi, suhteen nimi tai ominaisuuden nimi. Käytännössä kysely sisältää vakiot ja muuttujat. Ne kartoitetaan tietokannan sisältämiin vakioihin. Tällöin kyselyllä palautetaan kaikki löytyneet kartoitukset eli vastaavuudet graafihahmon ja graafitietokannan välillä [Angles et al., 2017]. Lisäksi hahmoon Q kuuluvat kaikki särmät kartoitetaan särmään alkuperäisessä graafitietokannassa G. Tällöin hahmon rakenne säilyy täsmäyksessä ja tulos on G:n aligraafi [Angles et al., 2017].

Graafihahmon täsmäyksellä vastataan kysymykseen, onko kyselynä syötettyä graafihahmoa vastaavaa aligraafia tietokannassa. Pelkän totuusarvon palauttamisen lisäksi täsmäyksellä palautetaan kyselyä vastaava aligraafi tai vastaavat aligraafit kyselyn tuloksena. Esimerkiksi graafihahmo, joka hakee ominaisuusgraafista kaksi henkilöä, jotka ovat kavereita keskenään, sisältää muuttujina entiteettityypin nimen henkilö, ja suhteen tyypin ”kaveri”. Kyselyllä haetaan ominaisuusgraafista kaikki entiteetit nimeltään ”kaveri”, josta lähtee kaari kaveri toiseen entiteettiin, jonka nimi on myös ”kaveri”. Kyselyn palauttaman tulos on kokoelma, joka koostuu pareista henkilöitä, jotka ovat kavereita keskenään. Graafihahmoa voi täydentää operaatiot, kuten projektio, yhdiste, tai suodatin. Näiden ominaisuuksien läsnä ollessa graafihahmoa kutsutaan kompleksiseksi graafihahmoksi [Angles et al., 2017].

Täsmäykselle ei ole olemassa välttämättä ainoastaan yhtä ainoaa määritelmää. Määritelmät voivat vaihdella ongelman mukaan [Gallagher, 2006]. Täsmäyksien määrittelemiseksi käytetään erilaisia semantiikkoja, joilla määritellään täsmäys tuottamaan oikeanlaiset tulokset [Angles et al., 2017]. Nämä semantiikat perustuvat homomorfismiin ja isomorfismiin [Angles et al., 2017]. Homomorfismiin perustuvat semantiikat eivät rajoita tuloksia ollenkaan, vaan palauttavat kaikki graafihahmoa vastaavat aligraafit graafista. Isomorfismiin perustuvat semantiikat sen sijaan rajoittavat tuloksia. Isomorfismia käyttävät semantiikat eivät toista eri arvoja sidottuna samaan muuttujaan. Niillä voidaan rajoittaa tuloksia myös solmu- tai kaarimuuttujien ehdoilla, niin että sama muuttuja ei voi olla sidottuna täsmäyksessä useaan hahmoon. Jokainen semantiikka sopii tiettyihin käyttötarkoituksiin, eikä ole olemassa yhtä semantiikkaa, joka sopii kaikkiin tarkoituksiin [Angles et al., 2017].

Pelkkä hahmon täsmäys ei riitä tarjoamaan vastauksia kyselyihin, koska graafitietokannat voivat olla sisältämältään datamäärältään valtavia kokonaisuuksia [Beaudou et al., 2019]. Graafihahmot eivät riitä tyydyttämään kyselytarpeita, koska graafihahmot kyselevät graafitietokantoja rajoitetulla tavalla antaen määrittää hahmoon yhden kaaren kerrallaan. Tämän puutteen ratkaisee graafitietokannoissa navigoinnilliset kyselyt. Ne mahdollistavat kyselyt, joissa kaaria pitkin voi navigoida mielivaltaisen matkan verran. Navigoinnillisilla kyselyillä ei tarvitse määrittää polkua solmujen välillä kaari kerrallaan [Angles et al., 2017]. Polun pituutta tarvitse rajoittaa, koska RPQ noudattaa refleksiivis-transitiivisuutta [Calvanese et al., 2003]. Poluilla tehtävät kyselyt ovat olleen navigoinnillisten kyselyiden ydin [Angles et al., 2017]. Polut on tunnistettu tutkijayhteisössä graafitietokannassa navigoinnin ytimeksi [Angles et al., 2017; Barcelo, 2013], koska niillä voidaan ilmaista saavutettavuutta, joka lukeutuu keskeisimpiin käsitteisiin graafeissa [Barcelo et al., 2014; Wood, 2012].

Tyypillisimpiin kyselyihin graafitietokannoissa lukeutuu kysely, joka tarkistaa polun olemassaolon kahden solmun välillä graafissa, jossa polku on kahden solmun välinen suunnattujen kaarien sekvenssi [Angles et al., 2017]. Jos graafitietokantaan on tallennettu esimerkiksi kaupunkien välisiä lentoyhteyksiä, voi olla hyödyksi selvittää ovatko kaksi kaupunkia yhteydessä. Navigoinnillisille kyselyille on hyödyksi asettaa ehtoja haettavalle polulle, kuten kaarien nimi ja niiden järjestys [Angles et al., 2017]. Tunnetuin ja tutkituin kieli, joka voidaan asettaa graafissa polulle ehtoja ovat RPQ:t.

Kuten graafihahmojen evaluoinnille, myös polkukyselyiden evaluoinnille on olemassa erilaisia semantiikkoja. Semantiikat määrittävät, miten polkukyselyn tuottama tulos määräytyy. Polkukyselyiden semantiikkoja on tutkittu erityisesti RPQ-mekanismien yhteydessä [Martens and Trautner, 2017]. Semantiikoista yleisimpiin kuuluu kahden solmun väliset kaikki polut palauttava semantiikka ja lyhimmän polun palauttava semantiikka. Needham [2019] toteaa lyhimmän polun etsimisen olevan yleisin tehtävä graafissa. Lisäksi yleisiä ovat semantiikat, jotka huomioivat tuloksessa sellaiset polut, joissa sama solmu tai sama kaari ei esiinny kahteen kertaan [Angles et al., 2017]. Kullakin semantiikalla on hyvät ja huonot puolensa. Kaikki polut huomioiva semantiikka on hyvä polun olemassaolon selvittämiseen, mutta tällä semantiikalla voi tuloksena olla ääretön määrä polkuja, jos polkuun kuuluu silmukoita [Angles et al., 2017]. Lyhin polku hakee aliluvussa 2.1 määritellyn lyhimmän mittaisen polun kahden solmun välille.

Semantiikoista laskennallisesti tehokkaimpia ovat minkä tahansa polun palauttaminen ja lyhimmän polun palauttaminen. Minkä tahansa polun palauttaminen voidaan tehdä lineaarisessa ajassa. Lyhimmän polun etsiminen graafista voidaan tehdä tunnettuja saavutettavuusalgoritmeja, kuten leveys- tai syvyyshakua, käyttäen. Näillä algoritmeilla on tunnettu tehokas aikavaatimus [Cormen, 2001]. Suoriin polkuihin perustuvat semantiikat sen sijaan ovat aikavaatimukseltaan NP-täydellisiä, joka tarkoittaa, että polynomisessa ajassa ei voida varmistaa onko vastaus oikea. Toisaalta nämä semantiikat ovat käytännöllisiä tapauksissa, jos halutaan välttää sellaisten polkujen palauttaminen tuloksissa, joissa ei ole samoja solmuja tai kaaria. Esimerkiksi sellaista lentoreittiä kahden lentokentän välillä ei ole järkevää näyttää, joka palaa reitin varrella olevalle lentokentälle, jossa on jo reitin aikana käyty. Erilaisista semantiikoista ja niihin liittyvistä ongelmista johtuen Martens ja Trautner [2017] mainitsevat RPQ-ilmausten evaluointi kuuluvan keskeisimpiin tutkimusongelmiin. Yhdistämällä graafihahmot ja polkukyselyt keskenään saadaan luotua navigoinnillisia graafihahmoja. Tällä tarkoitetaan graafihahmon kaarien nimeämistä RPQ:n periaatteiden mukaan [Angles et al., 2017]. Navigoinnillisilla graafihahmoilla voidaan selvittää kompleksisimpia suhteita graafitietokannasta. Useimpien modernien graafikyselykielten idea perustuu navigoinnillisiin graafihahmoihin.

Graafikyselykielten muita ominaispiirteitä ovat konjunktiiviset kyselyt, polkujen vertailu ja polkujen palauttaminen kyselyn tuloksena [Wood, 2012]. Graafitietokantojen yhteydessä konjunktiivisilla kyselyillä sidotaan kyselyyn useita polkuja. Konjunktiiviset kyselyt testaavat polkujen olemassaoloa graafissa, joiden kaikkien kyselyyn annettujen

polkujen täytyy löytyä, jotta konjunktiivinen kysely palauttaa totuusarvon tosi, ja aligraafin. Polkujen vertailuun ja polun palauttamiseen viitataan polkuun muuttujana [Wood, 2012]. Myös tulosjoukon aggregointi funktioilla on tärkeä ominaisuus graafikyselykielissä [Wood, 2012]. Aggregointifunktioista Wood [2012] mainitsee esimerkkinä tulosten kokonaislukumäärän, minimi- tai maksimiarvon, tai tulosten summan palauttavat funktiot. Näillä saadaan laskettua funktion mukainen tunnusluku kyselyn evaluoimasta tulosjoukosta.

3.3.2 Cypher

Tässä aliluvussa käsitellään graafikyselykieli Cypheriä [Cypher, 2020]. Cypher on deklaratiivinen kieli, jolla pystytään kyselemään ja manipuloimaan tietoa ominaisuusgraafeissa [Francis et al., 2018]. Alun perin Cypher on kehitetty kyselykieleksi Neo4J-graafitietokantaa varten, mutta Cypheriä käytetään nykyään kyselykielenä monissa kaupallisissa tietokannoissa ja tutkimuskäytössä [Francis et al., 2018].

Cypherin tietomalli perustuu ominaisuuksiin, rakenteeseen ja kompositioon [Cypher, 2020]. Cypherin syntaksi erottelee tietomalliin kuuluvat osat. Tietomalliin kuuluva kompositio koostuu listoista, ja kokoelmista, joilla avaimena on merkkijono [Cypher, 2020]. Ominaisuudet ovat arvoltaan Cypherin tukemia tietotyyppejä, kuten numeeriset tyypit, totuusarvo, merkkijonot sekä spatiaaliset- ja aika-arvot [Cypher, 2020]. Rakennetyyppejä ovat solmut, suhteet ja polut. Solmut ja suhteet koostuvat tunnisteesta, nimistä ja kokoelmasta ominaisuuksia [Cypher, 2020]. Cypher ilmaisee graafihahmon ASCII-taiteena [Francis et al., 2018]. Solmut kirjoitetaan sulkuparin sisälle, ja kaaret hakasulkeiden sisälle [Cypher, 2020]. Näin kyselyn syntaksi vastaa perinteistä graafin visualisointia, jossa solmut esitetään kaarevina muotoina ja kaaret näiden välisinä nuolina. Kaarta ympäröivät nuolen osat osoittavat mihin suuntaan kaarta pitkin kuljetaan. Sekä solmut, että kaaret sisältävät solmun tai kaaren nimen ja muuttujan nimen, jotta muuttujaan voidaan viitata myöhemmin kyselyssä. Nämä osat on eroteltu kaksoispistenotaatiolla, ja molemmat osat ovat valinnaisia. Solmu tai kaari voivat kyselyssä olla tyhjiä tai sisältää vain muuttujan. Näissä tapauksissa elementtien nimellä ei ole ehtoja, vaan se voi vastata mitä tahansa graafitietokannan elementtiä. Jos kaarelle ei ole asetettu ehtoja, hakasulkeet voi jättää tarpeettomana kyselystä pois ja kirjoittaa ASCII-nuolen haluttuun suuntaan suoraan solmusta toiseen solmuun. Jos kaaren suunnalle ei haluta asettaa ehtoja,

Cypherissä voi jättää kaaren kärkeä kuvaavan merkin kirjoittamatta, jolloin suhde ilmaistaan kahdella katkoviivalla [Cypher, 2020]. Elementit voivat sisältää vertailuoperaation, jolla verrataan elementin ominaisuuden yhtäsuuruutta suhteessa vakioon.

Cypherissä voidaan solmujen ja kaarten lisäksi nimetä polkuja. Cypherissä polku kirjoitetaan solmuista ja suhteista koostuvana vaihtelevana sekvenssinä. Polku määritellään kirjoittamalla [polun tunnus] = [solmujen ja kaarien sekvenssi]. Polun tunnus on uniikki muuttuja, johon voidaan viitata myöhemmin kyselyssä. Polun tunnusta voidaan esimerkiksi käyttää parametrina funktioille, joilla kuvataan polun osia yksityiskohtaisemmin. Esimerkkejä tällaisista funktioista Cypherissä on `relationships`, joka palauttaa kaikki polkuun kuuluvat kaaret. Funktio `nodes` puolestaan palauttaa kaikki polkuun kuuluvat solmut.

Cypher tukee RPQ-operaatioita rajoitetusti. Cypher mahdollistaa taulukon 1 RPQ-operaatioista operaatiot a, a^*, a^+, $a^{\{i,j\}}$ ja *****. Näiden operaatioiden vastine Cypherin syntaksissa sijoitetaan kaaren yhteyteen. Cypherissä voi navigoida kerrallaan yhtä kaaren nimeä kerrallaan tai asettaa sen villiksi kortiksi. Cypherin ilmaisut tukevat pituuspolkuja (*variable length path*), jotka ovat yksi RPQ-operaatioista. Pituuspolut mahdollistavat asettamaan polkuun suhteen, sekä minimi- ja maksimimäärän. Tällä täsmätään graafin polkuihin, joissa suhde esiintyy peräkkäin vähintään minimimäärän ja korkeintaan maksimimäärän. Esimerkkinä pituuspolusta on suhde `[:KNOWS*2..7]`. Tämä suhde täsmätään ominaisuusgraafissa sellaisiin polkuihin, joita kuljetaan solmusta toiseen solmuun vähintään kahden `KNOWS`-nimisen kaaren verran, mutta korkeintaan seitsemän kaaren verran [Angles et al., 2017]. Pituuspoluilla voidaan ilmaista myös tyhjät polut solmusta itseensä asettamalla minimimäärä nollaksi. Lisäksi Cypher voi katsoa tukevan RPQ:iden yhdistämistä ja konjunktiivisia kyselyitä.

Cypherissä voi asettaa graafihahmon kaarelle useita kaarien nimiä disjunktiota käyttäen. Disjunktio tuottaa onnistuneen täsmäyksen, jos mikä tahansa kaarista löytyy graafihahmossa määriteltyjen solmujen väliltä. Kyselykielen ominaisuuksista Cypher tukee projektiota, yhdistettä, erotusta, valinnaista täsmäystä ja suodatusta [Angles et al., 2017; Francis et al., 2018].

3.3.2.1 Semantiikka

Cypher käyttää semantiikkana kaari-isomorfismia sekä graafihahmojen täsmäämiselle että navigoinnillisissa kyselyissä. Kaari-isomorfismilla ei palauteta samaa kaarta useassa eri tuloksessa [Angles et al., 2017; Neo4J, 2020]. Navigoinnillisten kyselyiden yhteydessä Cypher-kyselyyn voi määritellä `shortestPath`-lausekkeella kyselyn hakemaan

nimenomaan kahden solmun välisen lyhimmän polun ja `allShortestPath`-lausekkeella voi huomioida kaikki kahden solmun väliset lyhimmät polut [Cypher, 2020; Angles et al., 2017].

3.3.2.2 Lausekkeet

Cypher-kysely koostuu lausekkeista [Francis et al., 2018]. Jokainen Cypherin lauseke on pohjimmiltaan funktio, joka saa syötteenä taulukon ja palauttaa taulukon. Koska jokainen funktio käyttää edellisen osan palauttamaa tulosta, koko kysely on kyselyyn kuuluvien funktioiden kompositio [Francis et al., 2018]. Näin ollen lausekkeiden järjestys vaikuttaa Cypher-kyselyn evaluointiin, vaikkakin väärässä järjestyksessä annettu lausekerakenne johtanee syntaksivirheeseen. Moni Cypherin lauseke muistuttaa SQL:n syntaksia. Holzschuher ja Peinl [2013] pitävät Cypheriä helposti opittavana SQL:n tunteville. Cypherin ja SQL:n samankaltaisuus onkin ollut Cypheriä kehitettäessä tarkoituksenmukaista helpottamaan käyttäjien siirtymistä SQL:stä Cypheriin [Francis et al., 2018].

Cypherissä kysely päättyy aina `RETURN`-lausekkeeseen [Francis, 2018]. Lauseke `RETURN` määrittelee kyselyn projektion [Francis et al., 2018], joka voi Cypherissä olla tyypiltään solmujen, kaarien tai ominaisuuksien joukko, alkeistyyppi tai polku. Paluuarvot voi sitoa parametrina johonkin funktioon, jolloin Cypher-kysely palauttaa funktion sen parametreilla tuottaman arvon. Lausekkeessa `RETURN` palautettavalle arvolle voi antaa alias-arvon `AS`-operaattorilla, joka näkyy tuloksena palautettavassa taulukossa [Cypher, 2020; Francis et al., 2018].

`MATCH` on Cypher-kyselyn lauseke, joka määrittää tietokannasta täsmättävän graafihahmon. Hahmo voi koostua poluista, solmuista ja kaarista, ja vertailuoperaatioista näihin kuuluvien ominaisuuksien suhteen. Graafihahmot saavat sisältää navigoinnillisia ominaisuuksia RPQ-mekanismia käyttäen, jolloin `MATCH`-lauseke tukee navigoinnillisia graafihahmoja. Hahmoja voi olla lausekkeessa yksi tai useita [Francis et al., 2018]. Mikäli `MATCH`-lausekkeessa on määritelty useita täsmättäviä hahmoja, ne erotellaan toisistaan pilkulla [Cypher, 2020]. Käytännössä yksi pilkuin eroteltu hahmo vastaa polkua, joka ei haaraudu. Tällöin jokaisella graafihahmoon kuuluvalla solmulla aste on korkeintaan kaksi. Useat graafihahmot voidaan ilmaista Cypher-kyselyssä myös usealla `MATCH`-lausekkeella. Cypherissä käytäntönä on, että toisiinsa yhteydessä olevat kuuluvat polut kirjoitetaan pilkulla eroteltuna yhden `MATCH`-lausekkeen sisään. Edellistä polkua seuraavat polut sisältävät viittauksen aiemmissa poluissa esiteltyihin muuttujiin. Hahmot, joiden ei oleteta olevan kiinni toisissaan, aloitetaan uudella `MATCH`-lausekkeella. Vaikka nämä

kaksi tapaa ketjuttaa useita hahmoja vaikuttavat samalta, näillä on semanttiset eronsa evaluoinnissa. Usea kyselyyn kuuluva MATCH-lauseke evaluoidaan toisistaan riippumatta. Koska Cypherin semantiikka graafihahmojen evaluoinnissa on toteutettu suhdeisomorfismilla, usean MATCH-lausekkeen sisältävän kyselyn evaluointi voi tuottaa enemmän tuloksia kuin yhden hahmon evaluointi. Useassa MATCH-lausekkeessa voidaan toisistaan riippumatta sisällyttää samoja osia graafitietokannasta kyselyn tuloksiin, mutta samassa hahmossa samoja osia ei sisällytetä kuin kerran tulokseen.

OPTIONAL MATCH-lauseke vastaa SQL:n ulkoista liitosta [Francis et al., 2018; Cypher, 2020]. Lausekkeessa korkeintaan täydennetään edeltäneen MATCH-lausekkeen graafihahmon määrittelyä. Mikäli tässä lausekkeessa annetulle hahmolle ei onnistuta muodostamaan täsmäystä tietokannasta, lauseke palauttaa totuusarvon sijaan tyhjän arvon eikä vaikuta koko kyselyn onnistumiseen.

UNWIND-lauseke on funktio, joka saa kyselyn aiemmista vaiheista tuotetun listan parametrina. Lauseke sitoo listan iteraattoriin, johon voidaan viitata kyselyn myöhemmissä osissa. Esimerkiksi Cypher-ilmaisu `UNWIND [1,2,3,4] AS rel` tuottaa iteraattorin `rel`, joka käy kaikki saamansa listan alkiot läpi, jos Cypher-muuttujaa `rel` kutsutaan kyselyssä [Cypher, 2020].

WITH-lauseke ketjuttaa kyselyn osat toisiinsa. Lausekkeessa manipuloidaan tulosjoukkoa ennen kuin se annetaan seuraaville kyselyn lausekkeille syötteenä [Cypher, 2020]. WITH-lausekkeella voidaan rajoittaa täsmäyksien lukumäärää antamalla yhdessä ORDER BY-lausekkeen kanssa kriteeri, jolla palautetaan ensimmäiset täsmäykset. WITH-lausekkeella voidaan luoda aggregaatteja, joita voidaan käyttää suodattamaan täsmäyksiä pois myöhemmin WHERE-lausekkeessa [Cypher, 2020].

WHERE-lauseke on suodatinfunktio, joka saa syötteenä edeltäneiden MATCH -, OPTIONAL MATCH tai UNWIND-lausekkeiden tuottamat taulukot. WHERE-lausekkeessa suodatetaan tuloksista pois ehtoja täyttämättömät täsmäykset. Koska WHERE-lausekkeen tehtävänä on eliminoida tuloksia, WHERE-lausekkeessa ei esitellä uusia muuttujia, vaan se tulee tehdä WHERE-lauseketta edeltävissä lausekkeissa. Sen sijaan WHERE-lausekkeessa voi esitellä muuttujia, joiden näkyvyys rajoittuu lausekkeen sisälle. Näitä muuttujia voi käyttää lausekkeen sisäisiin operaatioihin, kuten listojen iterointiin. WHERE-osaan voi ketjuttaa useita ehtoja käyttämällä Cypherin loogisia operaattoreita, kuten AND tai OR. Siinä missä MATCH-lausekkeessa vertailuoperaatioita voitiin tehdä vain ominaisuuksien

yhtäsuuruuden mukaan, WHERE-lausekkeessa tuloksia voi suodattaa lisäksi muiden vertailuoperaattorien mukaan, tai soveltaa suodattamiseen erilaisia funktioita [Cypher, 2020].

Muita oleellisia Cypherin lausekkeita ovat CREATE, DELETE ja SET, jotka päivittävät graafitietokantaa. Tietokantaa päivittäviä lausekkeita ei käsitellä tässä työssä.

3.3.2.3 Esimerkkikysely

Kyselyllä 2 havainnollistetaan Cypherin syntaksia. Kysely hakee tietokannasta polut, johon kuuluu entiteettityypin Person solmu, ja entiteettityypin Movie solmu. Näiden solmujen välillä kulkee suhdetta Directed kuvaava kaari, joka on suunnattu entiteettityypistä Person entiteettityyppiin Movie.

```
MATCH p0=(p:Person)-[d:DIRECTED]->(m:Movie)
WHERE m.name CONTAINS "int" RETURN p0
```

Kysely 2. Esimerkkikysely Cypherissä

Kyselyssä 2 (p:Person) ja (m:Movie) kuvaavat täsmättävän hahmon solmuja. Solmuissa p ja m ovat muuttujia, joilla viitataan kyselyssä annetun entiteettityypin mukaisiin solmuihin graafitietokannassa. Näihin kahteen muuttujaan kuuluvat entiteettityypit ovat Person ja Movie. Syntaksissa solmuja kuvaavien kaarisulkujen välissä oleva -[d:DIRECTED]-> kuvaa suhdetta. Suhde on suunnattuna solmusta Person solmuun Movie. Jotta kysely palauttaisi jotain, kyseltävässä graafitietokannassa täytyy olla sellainen aligraafi, johon kuuluu solmut entiteettityypiltään Person ja Movie. Näiden välillä täytyy kulkea suunnattu kaari nimeltään DIRECTED, joka on suunnattu Person entiteettiä kuvaavasta solmusta Movie entiteettiä kuvaavaan solmuun. Koko hahmo on sidottu polkuun tunnukseltaan p0. Polun tunnusta voi käyttää myöhemmin kyselyssä viittaamaan siihen aivan kuin mihin tahansa graafin elementtiin.

Täsmäysten jälkeen WHERE-lausekkeessa suodatetaan ehtoa tyydyttämättömät täsmäykset pois. Lausekkeessa esiintyy ehto, jossa m on viittaus MATCH-lausekkeessa esitellyn entiteettityypin Movie solmuihin liittyvä muuttuja. Muuttujasta pistenotaatiolla erotettu name on entiteetin ominaisuus. CONTAINS on yksi Cypherin merkkijonofunktioista ja ”int” arvo, jonka tulee esiintyä ominaisuuden name arvossa alimerkkijonona. Ehto siis suodattaa MATCH-lausekkeen tuottamasta täsmäyksestä pois kaikki ne täsmäykset, joilla ominaisuuden name merkkijonotyyppinen arvo ei sisällä merkkijonoa int.

Suodatettu taulukko siirtyy kyselyn seuraavalle lausekkeelle `RETURN`. `RETURN`-lausekkeessa kyselyn paluuarvoksi on määritelty polun tunnus `p0`. Kysely palauttaa kaikki hahmoon täsmäävät aligraafin solmut ja kaaret aligraafin mukaisessa rakenteessa. Tuloksista on suodatettu pois `WHERE`-lausekkeen ehtoja vastaamattomat tulokset.

4 XPathin graafitietokanta-semantiikan muodostaminen

Tässä luvussa esitetään XPathin semantiikka graafitietokannassa. Semantiikka eli merkitys muodostetaan tässä työssä kääntämällä XPath-ilmaisut Cypher-kielelle. Käännös täytyy tehdä, koska XPath ei sovellu suoraan kyselykieleksi Neo4J- graafitietokannalle. Tämän rajoituksen takia XPath käännetään Neo4J-tietokannan tukemalle Cypher-kyselykielelle. Käännöksessä käytetään XPathin versiota 1.0, ja Neo4J:n versiota 4.0, ja sen mukaista versiota Cypheristä. Vaikka XPath ja Cypher eroavat alkuperäisen kohdealueensa suhteen, kielillä on monia yhteneväisyyksiä navigaation ja hahmontäsmäämisen osalta. Tämän takia voi olettaa, että XPath-ilmaisut voidaan ilmaista Cypherillä, ja näin ollen käyttää XPathia Neo4J-tietokannan graafikyselykielenä. Aliluvussa 4.1 perehdytään aikaisempaan tutkimukseen XPathin käytöstä graafitietokannoissa. Aliluvussa 4.2 esitetään käännöksen rajaus ja periaatteet, joiden mukaan käännös XPathista tehdään Cypherille. Aliluku 4.3 esittää attribuuttikieliopin teorian. Attribuuttikielioppia käytetään muodostamaan semanttiset säännöt XPathilla Cypherille, ja aliluvusta 4.4 eteenpäin määritetään työssä käytetty attribuuttikielioppi. Aliluvussa 4.4 esitetään attribuuttikielioppiin kuuluvat kieltä jäsentävät symbolit, ja aliluvussa 4.5 semanttisten sääntöjen luomista varten käytetyt attribuutit. Aliluvussa 4.6 hyödynnetään aliluvuissa 4.4 ja 4.5 määriteltyjä symboleita ja attribuutteja luomaan työssä käytetyn attribuuttikieliopin säännöt.

4.1 Aikaisempi tutkimus

XPathin käyttöä on tutkittu graafitietokantojen kyselykielenä useasti 2010-luvulla. XPath on ollut useassa tutkimuksessa mittatikkukielenä graafitietokantojen vaatimuksia tutkittaessa [Libkin et al., 2013; Libkin et al., 2016]. Aiemmissa tutkimuksissa [Libkin et al., 2013; Kostylev et al., 2016] on perehdytty XPathin kaltaisen kielen semantiikkaan graafitietokannoissa teoreettisemmasta näkökulmasta. Tutkimukset ovat tarjonneet kattavan teoreettisen pohjan XPathin kaltaisten kielten käytölle graafitietokannoissa. Myös erilaisia käytännön kokeita on tehty XPathilla graafikyselykielenä [Cassidy, 2003].

Kiinnostusta XPathin käyttöön graafitietokannassa on herättänyt XPathin käyttötarkoitus, joka vastaa graafitietokannassa tehtäviä kyselyitä. Monessa lähteessä XPathin

on todettu toteuttavan hahmontäsmäystä samalla lailla kuin graafikyselykielet [Libkin et al., 2013]. Navigaation kannalta XPathin akselien tarjoamaa navigaatiota pidetään ilmaisuvoimaisena. Koska RPQ-mekanismi ottaa huomioon vain graafissa olevien suhteiden nimen, ovat Libkin ja muut [2016] ehdottaneet XPathin kaltaista kieltä graafikyselykieleksi. Libkinin ja muiden [2016] mukaan XPath pystyy syntaksissaan yhdistämään graafitietokannan sisältämän datan ja topologian. Libkin ja muut [2016] pitävät XPathia parhaana valintana pelkkiin graafin topologiaan perustuviin kyselyihin tehokkuutensa takia.

XPathilla navigoidaan puuympäristössä, jolloin sen käyttötarkoitusta voidaan sujuvasti laajentaa graafimaiseen ympäristöön [Libkin et al., 2013]. Libkin ja muut [2013] ovat todenneet, ettei siirtyminen käyttämään XPathia puuympäristöstä graafiympäristöön tuo tullessaan jyrkkää oppimiskäyrää. Libkin ja muut [2013] pitävät tuloksia XPathin kaltaisten kielten teoreettisesta käytöstä graafiympäristössä lupaavina. Syntaksin lisäksi XPathin kaltaista kieltä pystytään evaluoimaan tehokkaasti: parhaimmillaan lineaarisesti kielen navigoinnillisilla osilla [Libkin et al., 2013]. XPathin käyttöä graafitietokantoihin soveltuvana kyselykielenä on Libkinin ja muiden [2013] mukaan aliarvioitu.

4.2 XPathin tietomalli Cypherin tietomalliksi

Lähtökohtaisesti XPath on puumaisista XML-dokumenteista tiedonhakuun kehitetty kyselykieli. Tämän takia tulee tehdä joitain oletuksia, jotta Neo4J:n kaltaisen ominaisuusgraafitietokannan voi nähdä XPathiin sovellettavana rakenteena. Yhteistä XML-dokumenteille ja graafitietokannoille on kohdealueen entiteettien, niiden välisten suhteiden, ja entiteetteihin kuuluvien ominaisuuksien mallintaminen.

Kun XPathia vertaa Cypheriin, huomattavin ero on avainsanat. XPathissa kyselyn eri toiminnallisuudet ovat epäsuorasti tulkittavissa ilman avainsanoja. Cypherissä toiminnallisuudet on ryhmitelty avainsanoittain. Tästä huolimatta XPathin toiminnallisuudet voidaan ryhmitellä melko luonnollisesti Cypherin avainsanojen mukaan. Cypherin `MATCH`-lauseke vastaa XPath-ilmaisussa esiintyviä hahmoja. `WHERE`-lauseke sisältää pääasiassa XPath-ilmaisun predikaattien sisällön ja `RETURN`-lauseketta käytetään XPath-ilmaisun paluuarvon ilmaisemiseen. Työssä käytetään XPathin lähestymistapaa, jossa ilmaisun viimeinen askel määrittää kyselyn paluuarvon. On kuitenkin huomattavaa, että XPath-ilmaisua vastaavat Cypherin ilmaisut sijoitetaan `WHERE`- ja `MATCH`-osaan riippuen XPath-ilmaisusta. Myös lausekkeita `UNWIND` ja `WITH` käytetään ilmaisukohtaisesti. Nämä tilanteet käydään läpi myöhemmin aliluvussa 4.6. Aliluvussa 3.3.2.2 esitellyistä

lausekkeista käännöksessä ei käytetä lauseketta `OPTIONAL MATCH`, koska XPathissa ei ole olemassa ulkoista liitosta vastaavaa operaatiota.

XPathin tietomallin solmuista semantiikan muodostamiseen on valittu elementtisolmut ja attribuuttisolmut. XPathin elementtisolmut voi nähdä graafitietokannan solmuina, missä elementin nimi vastaa graafiin kuuluvan entiteettityypin nimeä. Valintaa voi perustella sillä, että XML:ssä elementit vastaavat mallinnettavan kohdealueen entiteettejä. XPathissa elementtisolmut kuvaavat näitä elementtejä. Graafitietokannoissa kohdealueen entiteettejä taas vastaavat solmut. Cypherissä solmu mallinnetaan kaarisulkeiden ympäröimäksi merkkijonoksi. Attribuuttisolmuilla taas kuvataan XML-dokumentin attribuutteja. Attribuutit ovat tärkeä osa kyselyä, koska varsinaisen rakenteen lisäksi mielekästä on ottaa huomioon rakenteeseen kuuluva data. Vaikka XPath-ilmaisussa attribuutit kirjoitetaan osaksi ilmaisua, niitä ei katsota elementtisolmuiksi XPathin tietomallissa. Tämä jako vastaa Cypherin tietomallia, jossa graafin rakenne eli solmut ja kaaret on erillään niiden ominaisuuksista. Myös attribuuttien esittäminen yhdistää XPathin ja Cypherin alkuperäisiä sovellusalueita. XML-dokumentissa entiteetteihin kuuluvat attribuutit ovat avain-arvo-pareja. Myös ominaisuusgraafeissa ominaisuudet mallinnetaan avain-arvo-pareina. Näin ollen XPathin attribuuttisolmun luontevana vastinparina voi pitää ominaisuusgraafin ominaisuutta. Attribuuttisolmu voidaan ilmaista XPathissa käyttämällä akselia *attribute* tai lyhennettyä muotoa @. Etenkin lyhennetty muoto auttaa erottelemaan datan XPath-kyselyn navigoinnillisista osista. Lyhennetty muoto onkin samalla työssä käytetty merkintätapa attribuuteille akselin koko nimen sijaan. XPathin attribuuttisolmujen sijoittaminen Cypher-kyselyyn on tapauskohtaista.

Muilla XPathin tietomallin solmutyypeillä ei ole olemassa vastinetta graafitietokannassa. Esimerkiksi juurisolmu on rajattu tämän työn käännöksestä pois. Kuten aliluvussa 2.1.1 todettiin, graafissa ei välttämättä ole sellaista juurisolmua kuin XML-dokumentissa on. XML-dokumentissa jokaisella dokumentilla esiintyy täsmälleen yksi juurielementti, jonka jälkeläisiä kaikki muut elementit ovat. Tämä johtaa myös siihen, että graafitietokannan yhteydessä täytyy käyttää XPath-ilmaisuissa suhteellista polkua absoluuttisen polun sijaan. Normaalisti XPathissa kysely aloitetaan joko //- tai /-notaatiolla. Graafitietokantaympäristössä kyselyn voi aloittaa vain //-notaatiolla, joka tarkoittaa navigoimisen aloittamista mistä solmusta tahansa. Koska tämä on graafitietokannoissa ainoa vaihtoehto kyselyn aloittamiselle, jätetään notaatio kirjoittamatta kyselyn alkuun tarpeettomana.

XPathin alkuperäisessä sovellusalueessa XML-dokumenteissa entiteettien välisiä suhteita kuvataan nimeämättöminä ja suunnattuina lapsi-vanhempi-suhteina. Suhteiden kautta voidaan edelleen määritellä transitiivisia esivanhempi- tai jälkeläinen-suhteita. Näiden suhteiden voi katsoa tuottavan suunnatun ja yhtenäisen graafin, jossa jokaisen graafiin kuuluvan kaaren maalisolmu on lapsisolmu ja lähtösolmuna on vanhempi-solmu. Näin XPathin alkuperäisestä sovellusalueesta saadaan graafimainen esitysmuoto. Graafitietokannoissa entiteettien välisiä suhteita mallinnetaan graafiin kuuluvina kaarina. Graafitietokannassa kaaret voidaan nimetä, ja niihin voi myös säilöä dataa. Graafitietokannassa elementtien jako solmuihin ja kaariin on selvä. Moderneissa graafikyselykielissä, kuten Cypherissä, niiden kuvaamiselle on omat syntaktiset sääntönsä. XPathin luontainen rajoitus on, että sen tarjoama navigaatio ei ota huomioon kaaren nimeä tai sen sisältämää dataa, koska näitä ei ole määritetty XML:n tietomallissa. Ominaisuusgraafia kyseltäessä hyödyllinen ominaisuus olisi kuitenkin ottaa navigaatiossa huomioon myös graafin sisältämät kaaret. Kyselyssä saatetaan hakea tietoa kaarista itsestään tai hakea tietyt ehdot täyttävä polku.

Käytettäessä XPathia ominaisuusgraafitietokantaan, valittuna lähestymistapana graafitietokannan sisältämät suhteet kuvataan XPathin tietomallissa elementtisolmuina. Lähestymistapa mahdollistaa kaaren nimen ja ominaisuuksien lisäämisen XPathin suhteille. Kaaret kuvataan XPath-ilmaisussa solmujen välissä. Tämä johtaa siihen, että XPathin tietomallin elementtisolmut jaetaan työssä kahteen luokkaan: solmuihin ja kaariin. Tällöin XPath-kyselystä tulee graafitietokannan kontekstissa solmujen ja kaarien sekvenssi. Työssä mahdolliset solmuista, kaarista ja attribuuteista koostuvat sekvenssit on esitetty taulukossa 3.

XPath-ilmaisu	**Ilmaisun solmujen, kaarien ja attribuuttien järjestys**
a/b/c	Solmu/Kaari/Solmu
a/b	Solmu/Kaari
a//c	Solmu//Solmu
polku/@d<suodatin>	Polku(*jokin aiemmista sekvensseistä*)/Attribuutti<suodatin>
@d<suodatin>	Attribuutti<suodatin>

Taulukko 3. XPathin elementit jaoteltuna solmuihin, kaariin ja attribuutteihin

Taulukossa 3 XPath-ilmaisun aloittaa solmu. Pääasiassa joka toinen XPath-ilmaisun solmu vastaa graafin solmua, ja joka toinen kyselyn solmu kaarta. Tämän säännön poikkeustapauksiin kuuluu transitiivinen sulkeuma käyttäen notaatiota ”//”, jonka molemmilla puolilla olevat solmut kuvaavat graafitietokannan solmuja. Tämä on valittu lä-

hestymistavaksi, sillä se on intuitiivinen tapa hakea mikä vain polku kahden solmun välille. Näin voidaan testata ovatko solmut yhdistetty toisiinsa, joka lukeutuu oleellisimpiin tehtäviin graafitietokannoissa. Poikkeustapauksiin kuuluu myös XPathin tietomalliin kuuluvat attribuuttisolmut. Attribuuttisolmu päättää aina XPath-ilmaisun. Attribuutti voi esiintyä solmun tai kaaren jälkeen, ja voi myös aloittaa XPath-ilmaisun. Attribuuttia voi XPathissa seurata suodatin, jolla suodatetaan annettua ehtoa vastaavat tulokset. Suodatinta käytettäessä ilmaisu palauttaa totuusarvotyyppisen arvon, joka vastaa kysymykseen löytyykö kyselyä vastaavia tuloksia graafitietokannasta. Solmujen ja kaarien yhteydessä suodattamiseen käytetään predikaatteja. Predikaatit voivat sisältää polkuja, attribuutteja tai attribuuttien vertailuja. Predikaattien semantiikka graafitietokannassa, ja niiden paikka Cypher-kyselyssä määräytyy tapauskohtaisesti.

Kun kaaret mallinnetaan elementtisolmuina, voi XPathilla navigoida ominaisuusgraafitietokannassa samaan tapaan kuin XML-dokumentissa. Valitussa lähestymistavassa XPathin akseleita voi käyttää ominaisuusgraafissa navigointiin XML-dokumentin tapaan. XPathin akseleilla voi ilmaista RPQ-operaatioita. Koska Cypherin navigaatio perustuu myös RPQ-mekanismiin, voidaan tällä tavalla XPathin ja Cypherin navigaatiolle muodostaa vastinparit. Nämä vastinparit on ilmaistu taulukossa 4. XPathissa akseli vastaa RPQ-operaattoria sekä solmutesti suhteen nimeä. XPathin semantiikassa graafitietokannassa akseleita käytetään pääasiassa kaariaskeleissa. XPathin solmutestin sisältö vastaa Cypher-kyselyssä suhteen nimeä. XPathin graafitietokantasemantiikan voi katsoa toteuttavan RPQ-mekanismin rajoitetusti siten, että navigointiin voi käyttää vain yhtä kaaren nimeä askelessa. XPathissa RPQ-periaatteiden kanssa yhteensopivia akseleita ovat *ancestor*, *ancestor-or-self*, *child*, *descendant* ja *descendant-or-self*, ja *parent*.

XPath-akseli *child* voidaan nähdä siirtymisenä yhtä kaarta sen suuntaisesti. *Descendant*-akselin merkitys graafitietokannassa on *child*-akselin transitiivinen sulkeuma. Akselilla navigoidaan kaaria niiden suuntaan, kunnes XPath-ilmaisussa akselin yhteydessä olevan solmutestin tai predikaatin ehto ei enää täyty. XPathin akselit tukevat myös navigaatiota lapsesta vanhempaan siirtymistä. Tämä tarkoittaa, että graafitietokannassa XPathilla pystytään ilmaisemaan navigointi kaarta käänteiseen suuntaan. Täten kaksisuuntainen RPQ on ilmaistavissa XPathissa. Kaksisuuntaisen navigaation graafissa mahdollistaa akselit *parent* ja *ancestor*. *Parent*-akseli toimii käänteisesti kuin *child*-akseli, jolloin navigoidaan yhtä kaarta sen käänteiseen suuntaan. Täten *ancestor*-akselille saadaan merkitys *parent*-akselin transitiivisena sulkeumana, jolla navigoidaan kaaria niiden

käänteiseen suuntaan, kunnes solmutestin tai predikaatin ehto ei enää täyty. Lisäksi akselit *descendant-or-self* ja *ancestor-or-self* navigoivat kuin akselit *descendant* ja *ancestor*, mutta palauttavat saavutettuna solmun myös sen solmun, josta navigoiminen alkaa.

RPQ	XPath	Cypher
a	child::a	`-[:a]->`
$\mathbf{a}^{+}$	descendant	`-[:a*]->`
$\mathbf{a}^{*}$	descendant-or-self	`-[:a*0..]->`
$\mathbf{a}^{-}$	parent::a	`<-[a]-`
$\mathbf{a}^{-+}$	ancestor::a	`<-[a*]-`
$\mathbf{a}^{-*}$	ancestor-or-self::a	`<-[a*0..]-`

Taulukko 4. Vastaavuudet RPQ-operaattoreiden, XPathin ja Cypherin ilmaisujen kesken

XPathin akseleista *following*, *following-sibling*, *namespace*, *preceding* ja *preceding-sibling* eivät ole mukana käännöksessä, koska niiden semantiikka graafitietokannassa on epäselvä. Työssä ei käytetä *self*-akselia eksplisiittisesti osana kyselyitä. Tällä saavutetaan kompaktimpi syntaksi, ja silti ymmärrettävä suhde solmujen välillä. XPathissa ilman akselia tapahtuva siirtyminen seuraavaan solmuun katsotaan oletuksena *child*-akselin mukaiseksi navigoinniksi. Samalla XPathin graafitietokantasemantiikassa tyhjä akseli katsotaan *child*-akseliksi ja tuottavan taulukon 4 vastaavan kaaren Cypher-ilmaisussa. Akselia voi käyttää myös XPathin solmuaskeleissa, jotta navigointisuunnan suhteen ei synny väärinkäsityksiä. Kuten myöhemmin aliluvun 4.6.4 säännöistä ilmenee, akselilla ei kuitenkaan muuteta solmuaskelen yhteydessä sen semantiikkaa.

XPathin tietomalliin kuuluvien polkujen voi katsoa vastaavan Cypherin tietomallin polkua. XPath-ilmaisussa polku alkaa ilmaisun ensimmäisestä solmusta ja päättyy viimeiseen solmuun. Poikkeustapauksiin kuuluu XPath-ilmaisun predikaatti, joka sisältää polun. Tällöin predikaattia edeltänyt polku ja predikaatin sisällä oleva polku katsotaan erillisiksi poluiksi. Poluilla on kuitenkin tässä tapauksessa yhteinen muuttuja, koska predikaattiin sidottu muuttuja kuuluu predikaatin ulkopuolella olevaan polkuun. Työssä on oletettu, että polku alkaa aina solmusta eli polun sisältävä predikaatti voidaan sijoittaa XPath-ilmaisussa ainoastaan solmun yhteyteen. XPath-ilmaisussa polun sisältävä predikaatti voi esiintyä lähes missä vain kyselyssä, myös predikaatin sisällä. Näin XPath-ilmaisussa pystytään mallintamaan haarautuvia polkurakenteita graafissa. Haarautuvilla rakenteilla tarkoitetaan usean polun löytämistä samasta alkusolmusta, joista jokaiselle täytyy löytyä täsmäys konjunktiivisten kyselyiden periaatteiden mukaisesti. Cypherissä haarautumista ei voi mallintaa näin sujuvasti. Sen sijaan polut tulee ilmaista Cypherin

syntaksissa peräkkäin polkujen sekvenssinä, jossa haarautuminen ilmaistaan viittaamalla aiemmin kyselyssä solmun yhteydessä esiteltyyn muuttujaan. Näin ollen työhön valitussa lähestymistavassa mallinnetaan XPathin polut sisältävät predikaatit polkujen sekvenssinä Cypherissä.

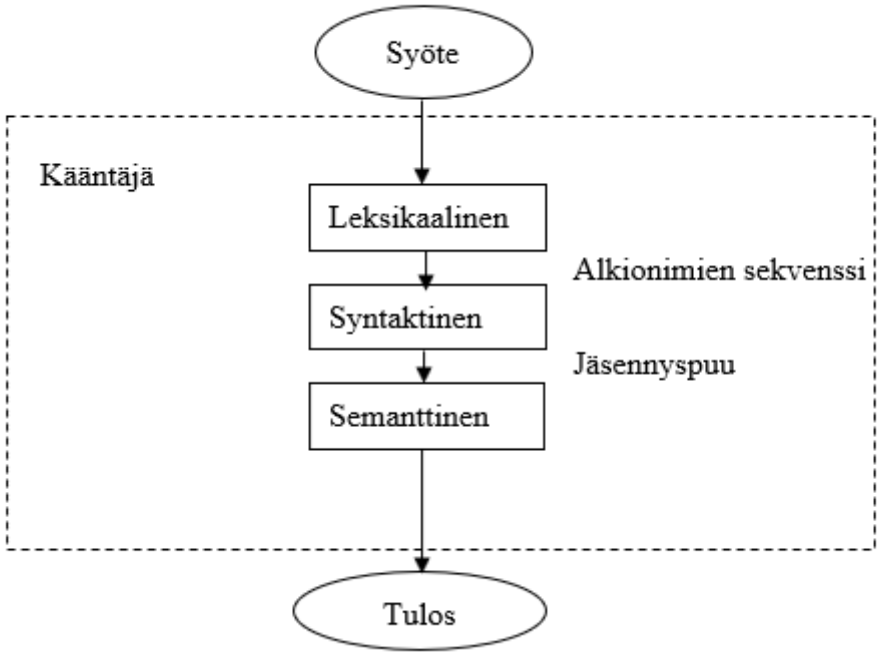
Pääasiassa työssä ei ole tarpeen käyttää polun tunnusta käännetyssä Cypher-kyselyssä. Poikkeustapauksena ovat sellaiset XPathin polut, joissa predikaatti on sidottu kaarisolmuun, ja kaaren akseli on transitiivinen. Tällöin kyselyssä siis halutaan asettaa ehto transitiivisille suhteille. Cypher ei salli muuttujien sijoittamista transitiivisten suhteiden yhteyteen, joihin voitaisiin viitata myöhemmin kyselyssä. Ratkaisuna on viitata tällaisissa tapauksissa transitiivisiin suhteisiin funktiolla `relationships`. Funktio ottaa parametrina polun tunnuksen, ja palauttaa kaikki polkuun kuuluvat suhteet. Tätä funktiota varten tarvitsee Cypherissä lisätä polun tunnus varsinaisen polun eteen. XPathissa tällaisessa tapauksessa predikaatti koskee vain siihen liitettyä elementtiä. Cypherissä funktio `relationships` palauttaa kuitenkin kaikki polkuun kuuluvat suhteet. Koska funktiota käytetään vain yhteen suhteeseen, täytyy polku katkaista, jotta suodatusta ei tehtäisi muiden polkuun kuuluvien suhteisiin nähden. Tällöin kaariaskelta edeltäneestä solmuaskelesta tehdään edellisen polun viimeinen solmu. Samalla solmuaskelen muuttuja liitetään osaksi uutta polkua. Mikäli kaariaskelta seuraavan solmuaskelen jälkeen predikaatissa esiintyy vielä lisää askelia, katsotaan kaariaskelta seuraava solmuaskel polun päättäväksi solmuksi. Tällöin uusi polku koostuu korkeintaan kahdesta solmusta ja yhdestä kaaresta.

XPathilla voi ketjuttaa useita polkukyselyitä loogisilla operaattoreilla. Tässä työssä käytetään vain loogista konjunktiota, jonka semantiikka on selvä Cypher-kyselyssä. Looginen konjunktio kirjoitetaan XPath-ilmaisussa predikaatin sisälle avainsanalla *and*. Cypherissä voi loogisella disjunktiolla mahdollista määrittää kyselyyn joukko kaaria. Näistä ainakin yhdellä täytyy löytyä kyselyn hahmoa tyydyttävä aligraafi, jotta kysely ei palauta tyhjää tulosjoukkoa. XPathissa disjunktion voi ilmaista epäsuorasti loogisella konnektiivilla predikaatin sisällä. Tässä tapauksessa ongelma on tällaisten kyselyiden syntaksi ja semantiikka Cypherissä. Tämän takia looginen disjunktio on jätetty pois työstä. XPath ei myöskään tue sellaisia RPQ-ilmaisuja, joilla on määritelty polkuun kuuluvien kaarten minimi- tai maksimimäärä. Cypherissä tämä voidaan ilmaista pituuspoluilla. Koska pituuspolkuja ei pystytä ilmaisemaan XPathissa, tämä ominaisuus on rajattu työstä pois. Lisäksi Cypherissä ei onnistu toteuttaa XPathissa ilmaistavaa järjestyksen mukaan palauttamisesta. Siinä missä XPathin alkuperäisessä sovellusalueessa XML:ssä

dokumentin rakenne koostuu järjestetystä kokoelmasta lapsisolmuja, vastaavanlaista järjestettyä rakennetta ei voida tunnistaa graafirakenteesta. Täten tämä XPathin ominaisuus on turha toteuttamiskelpoisuutensa vuoksi graafiympäristössä.

4.3 Kielioppi ja semantiikka

Tässä aliluvussa esitellään kielioppeihin ja semantiikkaan liittyvä teoria. Jokaisen ohjelmointikielen kääntäminen voidaan jakaa kolmeen vaiheeseen, joka on esitetty kuvassa 4. Kuvassa 4 näitä eri vaiheita kuvataan suorakulmiolla, kun taas kääntäjän syötteenä saama merkkijono ja kääntäjän tuottama merkkijono on kuvattu ellipsillä.



Kuva 4. Kielen käännösprosessi

Ensimmäisessä vaiheessa suoritetaan leksikaalianalyysi. Leksikaalianalyysissa erotellaan syötteenä saadusta merkkijonosta pienimmät tekstialkiot, jotka voidaan liittää johonkin alkionimeen [Harsu, 2005]. Tekstialkiot voidaan jaotella tunnuksiin, literaaleihin, operaattorisymboleihin tai erottimiin. Tekstialkiot ovat alkionimien ilmentymiä. Käytännössä ensimmäisessä vaiheessa tarkastetaan, että kieleen kuuluvat alkionimet sisältävät oikeanlaisen tekstialkion. Esimerkiksi, jos XPath-ilmaisussa esiintyy predikaatin avaava hakasulku, ilmaisussa täytyy esiintyä myös predikaatin sulkeva hakasulku. Muuten kysely on leksikaalisesti väärä, kääntäjä tuottaa virheilmoituksen, ja käännösprosessi keskeytyy. Leksikaalianalyysin toteutus perustuu säännöllisiin lausekkeisiin [Hopcroft and Ullman, 2001].

Kielen tekstialkioiden tunnistamisessa leksikaalianalyysissa käytetään usein kontekstivapaata kielioppia [Hopcroft and Ullman, 2001]. Kontekstivapaa kielioppi määritellään formaalisti monikkona G = (N, T, S, P). Monikossa N on nonterminaalien joukko, T on terminaalien joukko, P on kieliopin sääntöjen joukko, ja S on lähtösymboli, josta kielen jäsentäminen alkaa. Kontekstivapaassa kieliopissa pätee myös $N \cap T = \varnothing$, $p \in P$; $X_0 \rightarrow X_1 \ldots X_{np}$ ($X_0 \in N$, $X_i \in (N \cup T)$, $i \in \{1, \ldots, n\}$), $S \in N$ [Karol, 2016]. Toisin sanoen nonterminaalien ja terminaalien leikkaus on tyhjä joukko. Lisäksi jokaisella produktiolla pätee, että produktion vasemmalla puolella oleva symboli kuuluu nonterminaalien joukkoon. Oikealla puolella esiintyvät symbolit taas voivat kuulua nonterminaalien tai terminaalien joukkoon. Huomattavaa on, että kunkin säännön oikealla puolella olevien symbolien järjestys on merkitystä, joten säännön oikea puoli ei ole symmetrinen. Lähtösymboli S ei voi esiintyä yhdenkään säännön oikealla puolella [Knuth, 1968]. Tämä voisi johtaa sääntöjen läpikäymisessä päättymättömään silmukkaan.

Kontekstivapaassa kieliopissa terminaalisymbolit ovat leksikaalianalyysissa eroteltuja tekstialkioita. Nonterminaalisymbolit taas eivät esiinny syötteenä annetussa ohjelmassa. Ne kertovat missä järjestyksessä niiden kuvaama rakenne eli nonterminaalien ja terminaalien järjestys voi olla ohjelmassa. Nonterminaalien rakenteen määrittelee sääntö joukosta P. Nonterminaalisymbolille voi päteä kuitenkin kieliopissa useampia sääntöjä. Lisäksi lähtösymboli S kuuluu nonterminaalien joukkoon.

Syntaktisessa analyysissa leksikaalianalyysin tuottamaa alkionimisekvenssiä käyttäen rakennetaan jäsennyspuu. Kontekstivapaan kieliopin nonterminaalit muodostavat puun sisäsolmut, ja terminaalisymbolit puun lehtisolmut. Syntaktisesta analyysisista käytetään myös termiä jäsentäminen. Jäsennyksessä tarkastetaan kyselyn oikeellisuus sen sisältämien alkionimien järjestyksen suhteen. Vaikka kielen ilmaisu olisi leksikaalisesti oikein, voi kieli olla syntaktisesti väärä, toisin sanoen ilmaisua ei ole rakennettu kieliopin mukaan. Esimerkiksi XPathissa yhtäsuuruusvertailu täytyy toteuttaa siten, että yhtäsuuruusoperaattorin vasemmalla puolella esiintyy attribuutti tai elementti, ja oikealla puolella esiintyy arvo. Jos nämä olisivat yhtäsuuruusoperaattoriin nähden toisinpäin, syntaktisen tason mukaan kyseessä on virhe, ja ohjelman kääntäminen päättyy.

Kolmannessa vaiheessa suoritetaan semanttinen analyysi. Semanttinen analyysi saa syötteenä syntaktisen tason tuottaman jäsennyspuun ja tulokseksi tulee semantiikka. Esimerkiksi tässä työssä semantiikka on merkkijono-tyyppinen Cypher-kielen ilmaisu. Semantiikka muodostetaan kulkemalla jäsennyspuun kaikki solmut syvyyshaun mukaan.

Jokaisessa solmussa semantiikkaa täydennetään. Kun juurisolmuun on palattu, semantiikka on muodostettu ja kääntäminen valmis. Semanttiseen analyysiin käytetään usein työkaluna attribuuttikielioppia. Attribuuttikielioppi laajentaa kontekstivapaata kielioppia. Siinä kontekstivapaan kielen merkkijonoille voidaan muodostaa merkitys [Knuth, 1968]. Attribuuttikieliopissa määritellään kieliopin lisäksi attribuuttien ja semanttisten sääntöjen joukko [Knuth, 1968]. Formaalisti attribuuttikielioppi määritellään monikkona AG = (G, A, R). Monikossa G on kontekstivapaa kielioppi, A on käännöksessä käytettyjen attribuuttien äärellinen joukko, ja R on semanttisten sääntöjen joukko. Käytännössä attribuuttikieliopilla liitetään jokaisen kontekstivapaan kieliopin säännön yhteyteen semanttiset säännöt, joilla täydennetään muodostettavaa semantiikkaa. Attribuutti on objektin ominaisuus, joka voi saada arvon. Attribuuttikieliopissa attribuutit säilövät merkkijonon, jotka kuvaavat merkitystä. Attribuuttikieliopissa attribuutit voidaan edelleen jaotella periytyneiden ja synteettisten attribuuttien joukkoon [Knuth, 1968]. Kuvataan periytyneiden attribuuttien joukkoa tunnuksella INH ja synteettisten attribuuttien joukkoa tunnuksella SYN. Näillä joukoilla pätee $INH \cap SYN = \varnothing$ [Karol, 2016]. Attribuutti voi siis olla joko periytynyt tai synteettinen, muttei molempia. Periytyneet attribuutit kulkeutuvat jäsennyspuussa juurisäännöstä puun hierarkiassa alaspäin kohti puun lehtisolmuja. Tällä tavoin semanttisissa säännöissä voidaan käyttää hierarkiassa ylempänä määriteltyjä attribuuttien arvoja jäsennyspuussa hierarkiassa alempana olevien produktioiden semanttisissa säännöissä semantiikan muodostamiseen. Synteettinen attribuutti on attribuutti, joka arvotetaan kieliopin sääntöjen mukaan produktion semanttisissa säännöissä. Synteettisen attribuutin arvoa kuljetetaan jäsennyspuussa hierarkiassa ylöspäin kohti jäsennyspuun juurisolmua.

Attribuuttikieliopeille on olemassa erityistapauksia. Tällaisista erityistapauksista ovat esimerkiksi S- ja L-attribuoidut attribuuttikieliopit. S-attribuoidussa attribuuttikieliopissa voidaan käyttää ainoastaan synteettisiä attribuutteja. Periytyneet attribuutit eivät siis tällöin ole sallittuja. L-attribuoidut attribuuttikieliopit sallivat sekä periytyneet, että synteettiset attribuutit. L-attribuoidussa attribuuttikieliopissa synteettiset attribuutit voidaan arvottaa sekä periytyneillä, että synteettisillä attribuuteilla. L-attribuoidut attribuuttikieliopit ovat erikoistapauksia, koska säännön oikean puolen nonterminaalisymbolien periytyneet attribuutit voidaan arvottaa niiden vasemmalla puolella olleiden nonterminaalien synteettisillä attribuuteilla. Näin ollen jäsennyspuu käydään läpi käytännössä syvyyshaulla, koska jokaisen nonterminaalin periytyneiden attribuuttien arvottaminen odot-

taa sen vasemmanpuoleisten symbolien synteettisten attribuuttien arvottamista. Tämä tarkoittaa symbolin sisältämän puun läpikäymistä. Mikä tahansa oikean puolen symboli voidaan tietenkin arvottaa myös vanhempi-symbolilta periytyneiden attribuuttien arvoilla [Karol, 2016].

4.4 Symbolit

Tämän aliluvun aliluvuissa on esitelty käännöksessä käytetyt nonterminaali- ja terminaalisymbolit. Aliluvussa 4.4.1 esitellään XPathin kontekstivapaan kieliopin nonterminaalisymbolit, ja aliluvussa 4.4.2 terminaalisymbolit.

4.4.1 Nonterminaalit

Työssä XPath-kyselyn kontekstivapaan kieliopin luomiseen on käytetty 13 nonterminaalisymbolia. Taulukossa 5 kuvataan käännöksessä näitä symboleita.

Lyhenne	Nimi	Kuvaus
Q	Query	Kuvaa koko XPath-kyselyä
AX	Axis	Kuvaa askelen akselia
P	Pattern	Kuvaa kyselyyn kuuluvaa hahmoa
C	Node step	Kuvaa polkuun kuuluvaa solmuaskelta
E	Edge step	Kuvaa polkuun kuuluvaa kaariaskelta
N	Name	Kuvaa solmutestiä
PR	Predicate	Kuvaa hahmossa olevaa predikaattia
PRP	Predicate Pattern	Kuvaa predikaatin sisällä olevaa hahmoa
A	Attribute	Kuvaa kyselyyn kuuluvaa attribuuttia
SF	String function	Kuvaa merkkijonofunktiota
SFN	String function name	Kuvaa merkkijonofunktion nimeä
O	Operator	Kuvaa epäyhtäsuuruus- tai vertailuoperaattoria
V	Value	Kuvaa attribuutin arvoa

Taulukko 5. XPathin kontekstivapaan kieliopin produktiotyypit

4.4.2 Terminaalit

Työn terminaalisymboleiksi on määritelty 20 XPathin syntaksiin kuuluvaa merkkiä. Terminaalisymbolit on listattu tämän aliluvun taulukossa 6. Taulukon 6 terminaalisymbolien lisäksi terminaalisymboleihin kuuluu käytössä olevien akselien sekä funktioiden, solmujen, kaarien ja attribuuttien nimet. Myös yhtäsuuruusvertailuihin tai merkkijonofunktioihin kuuluvat numeeriset arvot sekä merkkijonot ovat myös terminaalisymboleja.

Symboli	Kuvaus
/	Erottaa askelet toisistaan
//	Erottaa askelet toisistaan
*	Villi kortti, jolla hyväksytään mikä tahansa merkkijono solmun tai kaaren nimeksi
..	Navigoi kaikkiin vanhempi-solmuihin. Lyhennetty muoto ilmaisulle parent::*.
@	Attribuutti-akselin lyhennetty muoto
::	Akselia ja solmutestiä erottava symboli
(	Funktion alussa oleva kaarisulku
)	Funktion lopussa oleva kaarisulku
[	Predikaatin alussa oleva hakasulku
]	Predikaatin lopussa oleva hakasulku
|	Yhdiste
=	Yhtäsuuruusoperaattori
>	Suurempi kuin-operaattori
<	Pienempi kuin-operaattori
>=	Suurempi tai yhtä suuri kuin-operaattori
<=	Pienempi tai yhtä suuri kuin-operaattori
!=	Erisuuri-operaattori
and	Konjunktio
”	Lainausmerkki, joilla ympäröidään merkkijono
,	Pilkku, jolla erotetaan merkkijonofunktion parametreja

Taulukko 6. Kontekstivapaan kieliopin terminaalisymbolit

4.5 Attribuutit

Käännöksessä on käytetty attribuutteja, joita kuvataan taulukossa 7. Taulukko 7 sisältää sekä synteettiset, että periytyneet attribuutit.

Nimi	Kuvaus
andInh	Periytynyt attribuutti, joka välittää jäsennyspuussa alaspäin merkkijonon AND tai tyhjän arvon. Riippumatta arvosta andInh sijoitetaan aina uuteen `WHERE`-lausekkeeseen lisättävän merkkijonon eteen.
andSyn	Synteettinen vastine andInh-attribuutille, jolla välitetään jäsennyspuussa ylöspäin `WHERE`-lausekkeen ehtojen eteen sijoitettava tyhjä arvo tai AND-merkkijono. Attribuutti andSyn saa arvokseen AND aina silloin, kun kyselyn `WHERE`-osaan sijoitetaan merkkijono, ja säilyttää arvon siihen asti, kun semanttinen analyysi on suoritettu.
appliesTo	Periytynyt attribuutti, joka sisältää Cypher-muuttujan tai tyhjän arvon, ja kertoo mihin muuttujaan predikaatti liittyy.
dotInh	Periytyneeseen attribuuttiin tallennetaan askeleeseen kuuluvia yhtäsuuruusvertailuja erottava pilkku.
dotSyn	Synteettiseen attribuuttiin tallennetaan askeleeseen kuuluvia yhtäsuuruusvertailuja erottava pilkku, joka lisätään Cypher-kyselyn `MATCH`-osaan.
equal	Synteettiseen attribuuttiin säilötään merkkijono, joka sijoitetaan `MATCH`-lausekkeessa solmun tai kaaren yhteyteen kuvaamaan yhtäsuuruutta ominaisuuden ja arvon välillä

insidePredicate	Periytynyt attribuutti, joka kuvaa totuusarvolla, mikäli jäsennyspuussa ollaan predikaatin sisällä.
pathInh	Periytynyt attribuutti saa arvokseen Cypher-kyselyyn mahdollisesti kuuluvan polun tunnuksen.
pathSyn	Synteettinen attribuutti saa arvokseen Cypher-kyselyyn mahdollisesti kuuluvan polun tunnuksen.
pathSeq	Synteettinen attribuutti, joka on tyhjä tai saa arvokseen merkkijonon pilkun, uuden polun nimen ja yhtäsuuruusmerkin. Attribuutti sijoitetaan `MATCH`-osaan hahmojen väliin, jos toinen hahmoista sisältää transitiivisen navigointisuunnan ja siihen liittyvän ehdon `WHERE`-lausekkeessa
pq	Synteettinen attribuutti, johon säilötään sellainen merkkijono, joka tulee Cypher-kyselyssä `MATCH`-lausekkeeseen. Attribuutin arvo sijoitetaan semanttisissa säännöissä aina ennen niin ikään `MATCH`-lausekkeeseen sijoitettavaa q-attribuutin sisältämää merkkijonoa, jos produktiossa asetetaan peräkkäin näiden attribuuttien sisältämät arvot.
q	Synteettinen attribuutti, johon säilötään `MATCH`-osaan sijoitettava kyselyn osa, ja joka kirjoitetaan pq-attribuutin läsnä ollessa sen perään.
ret	Synteettinen attribuutti, johon säilötään Cypher-kyselyn `RETURN`-osaan sijoitettava merkkijono eli Cypher-kyselyn palauttama muuttuja
transitiveInh	Periytynyt muuttuja, joka kertoo totuusarvolla, mikäli käytettävä navigaatiosuunta on transitiivinen.
transitiveSyn	Synteettinen muuttuja, joka kertoo totuusarvolla, mikäli käytettävä navigaatiosuunta on transitiivinen.
u	Synteettiseen attribuuttiin säilötään Cypher-kyselyn `UNWIND`-lausekkeeseen sijoitettava merkkijono
w	Synteettinen attribuutti, joka pitää sisällään `WHERE`-lausekkeeseen sijoitettavan Cypher-kyselyn osan.
with	Synteettinen attribuutti, joka pitää sisällään `WITH`-lausekkeeseen sijoitettavan merkkijonon

Taulukko 7. Semanttisissa säännöissä käytetyt attribuutit

4.6 Käännöksen kontekstivapaa kielioppi ja attribuuttikielioppi

Tässä aliluvussa esitetään työn käännöksen kontekstivapaa kielioppi ja attribuuttikielioppi. Aliluvuissa 4.6.1- 4.6.12 käydään läpi sääntöjä yksityiskohtaisemmin. Työn käännöstä varten luotu attribuuttikielioppi on esitetty liitteessä 1. Attribuuttikielioppiin kuuluvan kontekstivapaan kieliopin symboleita käsiteltiin aiemmin luvussa 4.4. Symboleista voidaan nyt muodostaa kontekstivapaa kielioppi $G_{PgXPath} = (NT, T, R, Q)$. Määritelmässä NT on aliluvussa 4.4.1 esiteltyjen nonterminaalisymbolien joukko {Q, AX, P, C, E, N, PR, PRP, A, SF, SFN, O, V}. Symbolilla T merkitään aliluvussa 4.4.2 esiteltyjen termi-

naalisymbolien joukkoa { /, //, *, .., @, ::, (,), [,], |, =, >, <, >=, <=, !=, and, ”, ’}. Määritelmässä lähtösymboliksi on määritelty Q eli koko XPath-ilmaisua kuvaava symboli. Symboli R vastaa liitteen 1 Sääntö-sarakkeen kokoelmaa säännöistä, joita ei niiden lukumäärän vuoksi kirjoiteta tähän.

Työssä käytetään attribuuttikielioppia määrittämään semanttiset säännöt käännökselle. Työn attribuuttikielioppi on L-attribuoitu attribuuttikielioppi. Attribuuttikielioppi voidaan esittää kolmikkona $AG_{Xypher} = (G_{PgXPath}, A, SR)$. Määritelmässä $G_{PgXPath}$ on aiemmin tässä luvussa määritelty kontekstivapaa kielioppi. A on aliluvussa 4.5 esitelty attribuuttien joukko {andInh, andSyn, appliesTo, dotInh, dotSyn, equal, path, pathSeq, pq, q, ret, transitiveInh, transitiveSyn, u, w}. Määritelmässä SR on liitteessä 1 esitetty kokoelma semanttisia sääntöjä. Työssä käytetty kontekstivapaa kielioppi $G_{PgXPath}$ eroaa XPathin version 1.0 [XPath, 1999] dokumentaatiosta. Kielioppi $G_{PgXPath}$ ottaa huomioon vain käännöksessä tarkastellut XPathin osat. Tämä on valittu lähestymistavaksi, koska jokaisen XPathin kieliopin sisältämän säännön huomioiminen tekisi attribuuttikieliopista sekavan ja pitkän kokonaisuuden. Koska jokaisella XPathin säännöllä ole välitöntä merkitystä tässä käännöstyössä, on tällaiset säännöt sivuutettu.

Työssä attribuuttikieliopin säännöt on yksilöity juoksevalla kokonaisluvulla luvusta yksi alkaen. Attribuuttikielioppiin kuuluvan kontekstivapaan kieliopin säännöissä nuolen vasemmalla puolella esiintyy säännön kuvaama nonterminaalisymboli, ja oikealla puolella sääntöön kuuluvien nonterminaali- ja terminaalisymbolien joukko. Attribuuttikieliopin semanttisissa säännöissä attribuuttia kuvataan funktionotaatiolla, jossa funktion nimenä on attribuutin nimi ja parametrina symbolin nimi. Attribuutin arvottaminen on kuvattu yhtäsuuruusmerkillä, jota seuraa attribuutille annettava arvo. Jokaisessa säännössä attribuutit on ryhmitelty synteettisiin ja periytyneisiin attribuutteihin. Synteettiset attribuutit ovat tunnistettavissa myös silloin, mikäli attribuutin funktionotaation parametrina oleva symboli vastaa kontekstivapaan kieliopissa säännön vasemmanpuoleista symbolia. Jokaisessa kontekstivapaan kieliopin säännössä terminaalisymboleihin kuuluvat solmujen, kaarien ja funktioiden nimet on kursivoitu. Muut terminaalisymbolit on lihavoitu. Lisäksi semanttisissa säännöissä on lihavoitu merkkijonot, jotka eivät ole Cypher-ilmaisun muuttujia vaan muita merkkejä, kuten kaari- ja hakasulut. Totuusarvot, joilla arvotetaan attribuutteja, mutta joita itsessään ei sijoiteta Cypher-kyselyyn, on alleviivattu. Kahden attribuutin arvojen peräkkäinasettelu (konkatenaatio) semanttisissa säännöissä

tapahtuu sijoittamalla kaksi attribuuttia peräkkäin eroteltuna välimerkillä. Joissain semanttisissa säännöissä ilmaistaan ehtoja if- ja else-ilmaisuilla. Nämä on erotettu pilkulla attribuutin varsinaisesta sisällöstä.

4.6.1 Kysely (Q)

Kielen aloitussymboli Q kuvaa koko XPath-kyselyä. Symboliin voidaan soveltaa neljää sääntöä. Kysely voi kuvautua hahmolle (sääntö 1), tai funktiolle (sääntö 2). Symboli voi kuvautua myös säännöille 3 ja 4, jolloin sääntö sisältää terminaalisymbolin | erottamaan hahmon tai funktion alikyselyistä.

1. Q → P
2. Q → F
3. Q1 → P | Q2
4. Q1 → F | Q2

Sääntöjen yhteydessä arvotetaan attribuutteja semanttisissa säännöissä. Periytyneet attribuutit *addInh, appliesTo* ja *finishPath* arvotetaan tyhjäksi jokaisessa säännössä 1-4. Periytynyt attribuutti *addInh* kuvaa tuleeko synteettisen attribuutin arvoon *w* lisätä Cypherin avainsana AND. Koska muodostettavassa Cypher-kyselyssä ei vielä ole sisältöä, attribuuttia *w* ei ole tässä vaiheessa tarpeen arvottaa. Myös periytynyt attribuutti *pathInh* arvotetaan tyhjäksi samasta syystä. Periytynyt attribuutti *finishPath* taas kuvaa totuusarvoa tuleeko `MATCH`-lausekkeeseen kirjoitettava polku katkaista pilkulla. Samasta syystä kuin periytyneillä attribuuteilla *pathInh* ja *andInh*, attribuutti arvotetaan tyhjäksi. Periytynyt attribuutti *insidePredicate* arvotetaan epätodeksi, koska kyselyn alussa ei olla predikaatin sisällä. Synteettiset attribuutit arvotetaan Cypher-ilmaisuilla symbolille Q kuvautuvissa säännöissä sen mukaan mitä rakenteita kyselystä on löytynyt. Säännön 1 tapauksessa attribuutti *ret* arvotetaan oheisen taulukon kuvaamalla tavalla.

Periytyneet attribuutit	*Synteettiset attribuutit*
andInh(P) = NULL	ret(Q) = **`MATCH`** pq(P) q(P)
appliesTo(P) = NULL	ret(Q) += **`UNWIND`** u(P), if u(P) ≠ NULL
finishPath(P) = false	ret(Q) += **`WITH`** with(P) , if withP) ≠ NULL
insidePredicate(P) = false	ret(Q) += **`WHERE`** w(P), if w(P) ≠ NULL
pathInh(P) = NULL	ret(Q) += **`RETURN`** ret(P)

Attribuutti *ret* saa arvokseen siis säännössä muodostetun Cypher-kyselyn. Cypher-kysely alkaa varatulla avainsanalla `MATCH`. Avainsanaa seuraa synteettisen attribuutin *pq*

arvo. Attribuuttia *pq* taas seuraa attribuutin *q* arvo. Molemmat attribuutit sisältävät `MATCH`-lausekkeeseen kuuluvan osan, mutta attribuutin *pq* arvo sijoitetaan aina ennen attribuutin *q* arvoa. Synteettisen attribuutin *ret* arvottaminen jatkuu Cypher-kyselyn valinnaisilla lausekkeilla. Tässä työssä vapaavalintaisiin lausekkeisiin kuuluvat `UNWIND`, `WITH` ja `WHERE`. Niihin liittyvät synteettiset attribuutit *u*, *with* ja *w*. Näistä lausekkeista kukin lisätään sisältöineen kyselyyn, mikäli niihin liittyvää synteettistä attribuuttia ei ole arvotettu tyhjäksi. Valinnaisilla lausekkeilla arvottamisen jälkeen synteettiseen attribuuttiin *ret* lisätään vielä kyselyn varattu avainsana `RETURN` ja siihen liittyvä kyselyn osa. Avainsana `RETURN` on pakollinen osa Cypher-kyselyä. Nyt synteettinen attribuutti *ret* on arvotettu, ja sen arvona on koko Cypher-kysely.

Säännöissä 2-4 attribuutti *ret* arvotetaan samankaltaisesti kuin säännössä 1. Säännössä 2 ja 4 attribuutti *ret* arvotetaan symbolilta F kuvautuneiden attribuuttien arvoilla. Säännöissä 3 ja 4 `RETURN`-avainsanaan kuuluvan kyselyn osan jälkeen attribuuttiin *ret* lisätään varattu avainsana `UNION` ketjuttamaan useita alikyselyitä. Avainsana `UNION` vastaa säännöissä 3 ja 4 esiintyvää XPathin terminaalisymbolia |. Molemmat näistä kuvaavat joukko-opillista yhdistettä. Yhdisteen jälkeen attribuuttiin *ret* lisätään alikyselylle Q2 arvotettu attribuutti *ret*, joka siis sisältää Cypher-kyselyn alikyselylle Q2.

4.6.2 Funktio (F)

Funktiosymbolilla kuvataan XPath-funktiota ja sitä merkitään työssä kirjaimella F. Tarkemmin kuvailtuna funktiosymbolilla F kuvataan XPathin aggregointifunktioita. Funktioon on sidottu yksittäinen hahmosymboli P. Funktio voi esiintyä työssä ainoastaan, kun se sisältää koko XPath-ilmaisun. Symbolia P ympäröi kaarisulkeet, jotka ovat terminaalisymboleja työn kontekstivapaassa kieliopissa. Tässä työssä funktion oikeanpuoleisen kaarisulkeen katsotaan päättävän XPath-kyselyn. Työssä käytetyt funktiot voivat palauttaa numeerisen arvon tai merkkijonon. Työssä käytetyt XPathin aggregointifunktiot on summan palauttava *sum*, keskiarvon laskeva *avg* ja kyselyn palauttamien tulosten määrän laskeva *count*. Funktiot on valittu vastaavuutensa takia, koska Cypherissä esiintyy samat funktioiden nimet. Ne myös evaluoidaan kielissä samalla tavalla. Keskiarvon palauttava *avg*-funktio ei virallisesti kuulu XPathin versioon 1.0, mutta sitä on helppo käyttää käännöksessä vastaavuutensa, koska Cypherissä on myös avg-funktio vastaavalla toiminnallisuudella.

5. F $\rightarrow$ *f_name*(P)

Säännön 5 semanttisiin sääntöihin liittyvät attribuutit ja niiden arvottaminen on kuvattu alla olevassa taulukossa. Verrattuna esimerkiksi sääntöön 1, symbolilla F kyselyn paluuarvo on numeerinen arvo.

Periytyneet attribuutit	*Synteettiset attribuutit*
andInh(P) = andInh(F)	q(F) = pq(P) q(P)
appliesTo(P) = appliesTo(F)	u(F) = u(P)
finishPath(P) = finishPath(F)	with(F) = with(P)
insidePredicate(P) = insidePredicate(F)	w(F) = w(P)
pathInh(P) = pathInh(F)	ret(F) = *f_name*(ret(P))

Symboli F esiintyy tässä työssä jäsennyspuussa välittömästi symbolin Q jälkeen. Tämän takia kaikki säännön periytyneet attribuutit välittävät saamansa arvot muuttumattomana symbolille P. Sama pätee myös synteettisiin attribuutteihin, jotka arvotetaan muuttumattomina symbolilta P. Poikkeuksena synteettinen attribuutti *q*, joka arvotetaan asettamalla peräkkäin `MATCH`-lausekkeeseen sijoitettavat synteettisten attribuuttien *pq* ja *q* arvot. Attribuutti *ret* arvotetaan kyselyyn kuuluvalla funktion nimellä, kaarisuluilla ja symbolin P synteettisen attribuutin *ret* sisältämällä merkkijonolla. Funktion nimi on merkkijono. Se siirtyy sellaisenaan XPath-ilmaisussa esiintyvästä funktion nimestä, koska käännökseen valittujen funktioiden nimet ovat samoja XPathissa ja Cypherissä. Funktion nimen jälkeen lisätään kaarisulut. Kaarisulkujen sisälle sijoitetaan symbolilta P kuvautunut synteettisen attribuutin *ret* arvo.

4.6.3 Hahmo (P)

Hahmo-symbolilla kuvataan XPath-ilmaisuun kuuluvia hahmoja. Hahmot ovat kyselyn osia, jotka sisältävät solmu- ja kaariaskeleita, attribuutteja tai hahmoja. Hahmo-symbolia kuvataan kirjaimella P. Symboli voi kuvautua kymmenelle vaihtoehtoiselle säännölle, joiden semantiikat ovat keskenään hyvin erilaisia Cypherissä huolimatta samanlaisesta syntaksista XPathissa.

Sääntö 6 kuvaa tilannetta, jossa XPath-hahmo sisältää kahden tai useamman solmun mittaisen polun. Tällöin hahmossa esiintyy vähintään kolme askelta: C, E ja P2. Symboli P2 sisältää vähintään yhden askeleen, koska kontekstivapaan kieliopin $G_{PgXPath}$ mukaan jokainen symboli P sisältää vähintään yhden solmuaskelen C. Symboli P voi tosin kuvautua säännölle P $\rightarrow$ A, jolloin P2 kuvautuisi attribuuttisolmulle. Koska P voi kuvautua myös säännölle P1 $\rightarrow$ P2/A, säännön 6 nonterminaali P2 kuvautuu luultavasti

sellaiselle hahmosäännölle, jossa esiintyy oikealla puolella nonterminaalisymbolina solmuaskel. Säännössä 6 ensimmäinen nonterminaali on solmuaskelta merkitsevä C. Sitä seuraa terminaalisymboli /, jonka jälkeen tulee nonterminaali kaariaskel E. Tätä seuraa jälleen terminalisymboli / ja edelleen hahmoa merkitsevä nonterminaalisymboli P2.

6. P1 → C / E / P2

Semanttisissa säännöissä arvotetaan periytyneitä ja synteettisiä attribuutteja. Synteettisistä attribuuteista *pq*, *q* ja *w* arvotetaan asettamalla peräkkäin vastaavat attribuutit säännön nonterminaalisymboleissa C, E ja P2. Lisäksi synteettisen attribuutin *pq* arvoon liitetään nonterminaalissa E arvotetun synteettisen attribuutin *pathSeq* merkkijono. Tämä merkkijono lisätään attribuuttiin *pq* kuvaamaan polun päättymistä ja uuden polun alkamista. Synteettinen attribuutti *pathSeq* voi olla tyhjä riippuen nonterminaalin E sisältämistä rakenteista, jolloin synteettisen attribuutin *pq* osaksi liitetään tyhjä arvo. Synteettiset attribuutit *ret, u* ja *with* arvotetaan nonterminaali P2:ssa arvotetulla attribuutin *ret* arvolla. Synteettinen attribuutti *ret* saa arvonsa symbolilta P2, koska XPath-ilmaisun paluuarvo on ilmaisun oikeanpuoleisin elementti. Synteettiset attribuutit *u* ja *with* arvotetaan myös suoraan symbolilta P2 kuvautuneelta symbolilta. Nämä attribuutit arvotetaan attribuuttikielopissa vain säännössä 7, jolle säännössä 6 ainoastaan symboli P2 voi kuvautua.

Periytyneistä attribuuteista periytynyt attribuutti *pathInh* arvotetaan symbolilta E kuvautuneen synteettisen attribuutin *pathSyn* arvolla. Periytynyt attribuutti *appliesTo* saa arvokseen symbolin C attribuutin *ret* arvon siltä varalta, jos symbolin E säännössä tarvitaan edeltäneen solmuaskelen tunnusta. Periytynyt attribuutti *finishPath* arvotetaan totuusarvolla, joka arvotetaan symbolin E säännöissä. Attribuutti *finishPath* välitetään symbolille P2 välittämään tiedon siitä kuuluuko muodostettavassa Cypher-ilmaisussa polku lopettaa hahmon yhteydessä. Tämä tarkoittaa sitä, että jos hahmon yhteydessä on käytetty jotain polkutunnusta, voidaan polku määrittää päättymään. Tällöin myöhemmin Cypher-kyselyyn lisättävät elementit eivät ole sidottuna samaan polkutunnukseen. Periytynyt attribuutti *andInh* arvotetaan aina edeltäneen nonterminaalisymbolilta kuvautuneen synteettisen attribuutin *andSyn* arvolla. Esimerkiksi nonterminaalisymboli E saa attribuutin *andInh* arvoksi sitä edeltäneen symbolin C semanttisissa säännöissä arvotetun synteettisen attribuutin *andSyn* arvon. Näin toimimalla tieto avainsanan AND lisäyksestä Cypher-kyselyyn välittyy kaikkialle jäsennyspuuhun. Säännön oikean puolen ensimmäisen symbolin C attribuutin arvottaminen poikkeaa, koska periytynyt attribuutin *andInh* arvo tulee suoraan symbolilta P1 periytyneeltä vastaavalta attribuutilta. Symbolit C, E ja P2 arvote-

taan periytyneen attribuutin *insidePredicate* arvolla riippuen siitä, onko hahmo predikaatin sisällä. Lisäksi tieto transitiivisesta kaariaskelen akselista välittyy kaariaskelta E symbolille P2 periytyneellä attribuutilla *transitiveInh*.

Periytyneet attribuutit	*Synteettiset attribuutit*
andInh(C) = andInh(P1)	pq(P1) = pq(C) pathSeq(E) pq(E) pq(P2)
finishPath(C) = finishPath(P1)	q(P1) = q(C) q(E) q(P2)
andInh(E) = andSyn(C)	w(P1) = w(C) w(E) w(P2)
appliesTo(E) = ret(C)	ret(P1) = ret(P2)
appliesTo(P2) = ret(E)	with(P1) = with(P2)
andInh(P2) = andSyn(C)	u(P1) = u(P2)
finishPath(P2) = finishPath(E)	
pathInh(P2) = pathSyn(E)	
insidePredicate(C) = insidePredicate(P1)	
insidePredicate(E) = insidePredicate(P1)	
insidePredicate(P2) = insidePredicate(P1)	
transitiveInh(P2) = transitiveSyn(E)	

Säännössä 7 esiintyy kaksi askelta. Sääntö kuvaa hahmoa, jolle on määritelty täsmälleen yksi solmuaskel ja yksi kaariaskel eikä hahmo jatku kaariaskelen jälkeen. Solmuaskelta kuvataan symbolilla C ja kaariaskelta symbolilla E. Solmuaskelta ja kaariaskelta erottaa terminaalisymboli /.

7. P → C / E

Semanttisissa säännöissä attribuutit *pq*, *q*, *w*, *pathInh*, *andInh* ja *insidePredicate* arvotetaan pääasiassa samalla lailla kuin säännössä 6. Näitä arvottamisia ei käsitellä uudestaan tässä yhteydessä. Poikkeuksena synteettisen attribuutin *pq* arvottamisen, jossa attribuutin arvon perään liitetään sulkupari () ilman sisältöä. Sulkuparilla kuvataan solmua, sillä Cypher-kyselyssä hahmon täytyy päättyä solmuun. Sääntöön liittyvä XPath-kysely taas päättyy kaariaskeleeseen. Säännössä synteettisen attribuutin *ret* arvottaminen tapahtuu symbolilta E kuvautuneen synteettisen attribuutin *pathSyn* mukaan. Jos attribuutti *pathSyn* on arvotettu, attribuutti *ret* saa arvokseen Cypherin funktion `relationships`. Funktion saa parametrina synteettisen attribuutin *pathSyn* sisältämän merkkijonon, joka kuvaa polun tunnusta. Synteettinen attribuutti *pathSyn* on käytännössä arvotettu, mikäli symbolin E kuvaaman kaariaskelen akseli on transitiivinen. Tällöin Cypherin

sääntöjen mukaan ei ole mahdollista lisätä muuttujan nimeä kaaren yhteyteen. Mutta koska säännön viimeinen askel on kaariaskel, täytyy tämä ilmaista kyselyn paluuarvona. Tähän käytetään Cypherin funktiota `relationships`, jolla pystytään palauttamaan kaaret käyttäen polun tunnusta funktion parametrina. Jos symbolin E rakenteesta ei löydy transitiivista, symbolilta E kuvautuneen synteettisen attribuutin *ret* arvo kuvautuu myös symbolin P synteettisen attribuutin *ret* arvoksi.

Periytyneet attribuutit	*Synteettiset attribuutit*
finishPath(C) = finishPath(P1)	q(P) = q(C) q(E)
andInh(C) = andInh(P)	pq(P) = pq(C) pathSeq(E) pq(E) ()
pathInh(C) = pathInh(P)	w(P) = w(C) w(E)
pathInh(E) = pathInh(P)	pathSyn(P) = pathSyn(E)
appliesTo(E) = ret(C)	ret(P) = **relationships**(pathSyn(E)) , if pathSyn(E) ≠ NULL
andInh(E) = andSyn(C)	ret(P) = ret(E) else
insidePredicate(C) = insidePredicate(P)	
insidePredicate(E) = insidePredicate(P)	

Säännössä 8 hahmosymbolin säännön oikea puoli koostuu vain yhdestä nonterminaalisymbolista, joka on solmuaskel C. Tällöin hahmossa ei esiinny kaariaskelia.

8. P → C

Semanttisissa säännöissä kukin synteettisistä attribuuteista arvotetaan suoraan niillä arvoilla, jotka kuvautuvat symbolilta C. Periytyneet attribuutit *pathInh* ja *andInh* vastaavasti perivät arvonsa suoraan symbolilta P.

Periytyneet attribuutit	*Synteettiset attribuutit*
pathInh(C) = pathInh(P)	q(P) = q(C)
andInh(C) = andInh(P)	pq(P) = pq(C)
insidePredicate(C) = insidePredicate(P)	w(P) = w(C)
	with(P) = with(C)
	u(P) = u(C)
	ret(P) = ret(C)
	andSyn(P) = andSyn(C)

Säännössä 9 esiintyy kaksi nonterminaalisymbolia C ja P2. Niitä erottaa terminaalisymboli //. XPathissa notaatio // polun keskellä tarkoittaa navigoimista kahden elementin välillä ottamatta kantaa elementtien välisen polun pituuteen tai rakenteeseen. XPathin

semantiikassa graafitietokannassa sääntö kuvaa navigoimista graafissa solmujen välillä suunnattua polkua pitkin. Tämä pätee tämän työn kontekstivapaassa kieliopissa, sillä hahmoa kuvaava symboli P2 on määritelty alkamaan aina solmuaskelella C. Navigaatio graafitietokantaan kuuluvista solmuista solmuihin ei tällöin ota kantaa polun pituuteen tai siihen kuuluviin suhteisiin.

9. P1 → C**//**P2

Säännön semanttisissa säännöissä synteettiset attribuutit *q*, *w, with, u* ja *ret* arvotetaan kuten säännössä 6. Attribuutit *pq*, *q* ja *w* arvotetaan asettamalla kunkin attribuutin arvot symboleilta C ja P2 peräkkäin. Arvotettaessa attribuuttia *pq*, sijoitetaan symbolien C ja P2 palauttamien synteettisen attribuutin pq arvojen väliin notaatio –[*]->. Tämä notaatio vastaa Cypherissä XPathin //-symbolia, ja kuvaa samalla XPathin semantiikkaa vastaavaa navigointia graafissa. Hahmon paluuarvoksi attribuuttiin *ret* arvotetaan symbolilta P2 saatu attribuutin *ret* arvo, koska se on kyselyssä oikeanpuolisimpana.

Periytyneet attribuutit	*Synteettiset attribuutit*
finishPath(C) = finishPath(P1)	pq(P1) = pq(C)**-[*]->**pq(P2)
pathInh(C) = pathInh(P1)	q(P1) = q(P) q(P2)
pathInh(P2) = pathInh(P1)	u(P1) = u(P2)
andInh(P2) = andSyn(C)	with(P1) = with(P2)
insidePredicate(C) = insidePredicate(P1)	w(P1) = w(C) w(P2)
insidePredicate(P2) = insidePredicate(P1)	ret(P1) = ret(P2)

Säännössä 10 hahmo sisältää kaksi nonterminaalisymbolia P2 ja A. Niitä erottaa terminaalisymboli /. Sääntö kuvaa mitä tahansa XPath-hahmoa, joka päättyy attribuuttisolmuun. Säännössä täytyy olla attribuuttisolmun lisäksi hahmo P2. Tilanne, jossa attribuuttisolmu esiintyy ainoana osana hahmoa, esiintyy säännössä 11.

10. P1 → P2**/**A

Säännön 10 semanttisissa säännöissä suurin osa synteettisistä attribuuteista kuvautuvat suoraan symbolilta P2. Tällaisia attribuutteja ovat *q*, *pq* ja *w*. Lisäksi periytyneiden attribuuttien *pathInh* ja *andInh* arvot siirtyvät suoraan symbolilta P1 symbolille P2. Semanttisissa säännöissä esiintyy säännön sisäiseen käyttöön tarkoitettu muuttuja.

Sääntö 10 on sääntöjen 11 ja 12 lisäksi attribuuttikieliopin AG_{Xypher} ainoa sääntö, jossa arvotetaan synteettiset attribuutit *u* ja *with*. Näitä attribuutteja ei arvoteta säännössä kuitenkaan aina, vaan tapauskohtaisesti. Tällainen tapaus on, kun symbolia A edeltäneen hahmon viimeisenä symbolina esiintyy kaariaskel, jolla on transitiivinen akseli. Tämä

ilmenee semanttisissa säännöissä, kun symbolilta P2 kuvautuneella synteettisellä attribuutilla *pathSyn* ei ole tyhjää arvoa. Tällöin synteettinen attribuutti *u* arvotetaan Cypherin funktiolla `relationships`. Funktion parametriksi sijoitetaan symbolilta P2 kuvautuneen synteettisen attribuutin *pathSyn* arvo. Funktion jälkeen synteettisen attribuuttiin lisätään AS-operaattori, ja sitä seuraa uniikki Cypher-muuttuja. Muuttujaa kuvataan säännössä nimellä *var*. Nyt uusi muuttuja viittaa funktiolla luotuun iteraattoriin. Synteettisen attribuutin *u* arvo sijoitetaan lopullisessa Cypher-kyselyssä `UNWIND`-lausekkeeseen. Kuten säännössä 7 todettiin, transitiiviselle suhteelle ei Cypher-ilmaisussa ole mahdollista määrittää muuttujaa. Viittaus täytyy tehdä hyödyntäen `relationships`-funktiota ja siihen liittyvää tunnusta. Koska `UNWIND` tuottaa vain iteraattorin, käytetään `WITH`-lauseketta viittaamaan listan alkioihin kyselyssä myöhemmin. Lausekkeen arvottamiseen käytetään synteettiseen *u* attribuuttiin lisättyä uniikkia tunnusta. Tunnuksen perään lisätään synteettisen attribuutin *u* tavoin `AS`-operaattori, ja sitä seuraa `WITH`-lauseketta varten luotu uusi tunnus. `WITH`-lausekkeen sisältö sijoitetaan synteettiseen attribuuttiin *with*.

Symbolilta P2 kuvautuneen synteettisen attribuutin *pathSyn* arvo määrää myös synteettisen attribuutin ret arvottamisen. Jos synteettinen attribuutin *pathSyn* on arvotettu, lisätään synteettisen attribuutin *ret* arvoon sama tunnus, joka lisättiin synteettiseen attribuuttiin *with*. Jos attribuutti *pathSyn* on tyhjä, attribuutti *ret* arvotetaan symbolilta P2 kuvautuneen synteettisen attribuutin *ret* arvolla. Molemmissa tapauksissa lisättyä arvoa seuraa pisteellä eroteltuna symbolilta A kuvautunut synteettisen attribuutin *ret* arvo. Säännössä 10 asetetaan Cypher-kyselyn paluuarvoksi kyselyn palauttamien solmujen tai kaarien arvojen joukko XPath-kyselyssä annetulla attribuutilla.

Jos sääntöä sovelletaan predikaatin sisällä, säännössä arvotetaan myös synteettinen attribuutti *w*. Predikaatin sisällä sääntöä ei käytetä määrittämään koko kyselyn paluuarvoa, sillä predikaatissa ei määritetä XPathissa paluuarvoa. Sen sijaan sääntöä käytetään tarkistamaan, onko XPath-ilmaisussa attribuuttisolmua edeltäneellä askelella symbolin A kuvaamaa attribuuttia ominaisuutena. Olemassaolon tarkistaminen ilmaistaan Cypherissä avainsanalla `EXISTS`. Avainsanaa seuraa tunnus ja attribuutin nimi. Nämä on eroteltu toisistaan pisteellä, ja ympäröity kaarisulkeilla. Tunnus riippuu tässäkin tapauksessa siitä, onko symbolilta P2 kuvautuneen synteettisen attribuutin *pathSyn* arvo tyhjä vai ei.

Periytyneet attribuutit	*Synteettiset attribuutit*
pathInh(P2) = pathInh(P1) andInh(P2) = andInh(P1) transitiveInh(P2) = transitiveSyn(P2) insidePredicate(P2) = insidePredicate(P1)	q(P1) = q(P2) pq(P1) = pq(P2) var = new_v() rel = new_v() u(P1) = **relationships(** pathSyn(P2) **) AS** var , if pathSyn(P2) ≠ NULL with(P1) = var **as** rel, if pathSyn(P2) ≠ NULL w(P1) = w(P2) , if insidePredicate w(P1) = w(P2) andSyn(P2) **EXISTS**(rel.ret(A)) if pathSyn(P2) ≠ NULL && insidePredicate w(P1) = w(P2) andSyn(P2) **EXISTS**(ret(P2).ret(A)) , if pathSyn(P2) == NULL && insidePredicate ret(P1) = rel.ret(A) if pathSyn(P2) ≠ NULL ret(P1) = ret(P2) **.** ret(a) else

Hahmon lopussa olevaan attribuuttiin voi soveltaa arvon vertailua. Vertailu suoritetaan XPath-ilmaisussa aliluvussa esitellyillä XPathin operaattoreilla 4.6.11. Nämä tilanteet on kuvattu säännöissä 11 ja 12. Kun polun päätteenä tehdään attribuutin arvon vertailua, polun P1 paluuarvon tyyppi on totuusarvo. Tämä vastaa XPathin luonnollista evaluointia hierarkkisissa dokumenteissa. Cypherissä totuusarvotyyppisen paluuarvon palauttaminen vaatii kiertotien, koska Cypherin syntaksissa ei ole suoraa tapaa ilmaista totuusarvotyyppistä paluuarvoa. Työhön valitussa lähestymistavassa sijoitetaan semanttisissa säännöissä synteettiseen attribuuttiin *ret* merkkijono `COUNT(muuttuja) > 0`. Aggregointifunktio `COUNT` laskee kyselyn palauttaman tulosten lukumäärän. Mikäli niitä on enemmän kuin nolla, palauttaa funktio totuusarvon tosi. Muussa tapauksessa funktion palauttama arvo on epätosi. Tunnuksella viitataan siihen muuttujaan, joka olisi palautettu Cypher-kyselystä ilman attribuutin arvon vertailua. Tunnus riippuu tässäkin säännössä symbolin P2 kuvautuneen synteettisen attribuutin *ret* arvosta. Koska sen arvottamisen periaatteet käytiin läpi säännössä 10, niitä ei kerrata tässä yhteydessä uudestaan.

Itse suodattaminen sijoitetaan Cypher-kyselyn lausekkeeseen `WHERE`. Näin ollen säännöissä arvotetaan synteettinen attribuutti *w*. Jälleen kerran symbolilta P2 kuvautuneen attribuutin *ret* arvo ratkaisee arvottamisen periaatteet. Jos attribuutilla on arvo, arvotetaan synteettinen attribuutti *w* käyttämällä jälleen funktiota `relationships`. Parametriksi funktio saa muuttujan *rel* arvon. Muuttuja *rel* on jälleen `WITH`-lausekkeeseen sijoitettava uniikki tunnus, kuten säännössä 10. Lausekkeeseen `WHERE` sijoitettaessa lisätään funktion eteen Cypherin avainsana `ALL` kuvaamaan vaatimusta, että kaikkien polkuun kuuluvien suhteiden tulee täyttää ehto. Mikäli synteettistä attribuuttia *pathSyn* ei ole arvotettu, lisätään attribuuttiin *w* symbolilta P2 kuvautunut synteettisen attribuutin *ret* arvo pisteellä eroteltuna symbolilta A kuvautuneen attribuutin *ret* arvosta.

11. P1 $\rightarrow$ P2/A=V

12. P1 $\rightarrow$ P2/A O V

Alla olevassa taulukossa on esitelty säännön 12 semanttiset säännöt. Säännöt vastaavat säännön 11 semanttisia sääntöjä ainoana erona, että w(O) on korvattu yhtäsuuruusmerkillä.

Periytyneet attribuutit	*Synteettiset attribuutit*
pathInh(P2) = pathInh(P1) andInh(P2) = andInh(P1) transitiveInh(P2) = transitiveSyn(P1) insidePredicate(P2) = insidePredicate(P1)	q(P1) = q(P2) pq(P1) = pq(P2) var = new_v() rel = new_v() u(P1) = **relationships(** pathSyn(P2) **) AS** var , if pathSyn(P2) ≠ NULL with(P1) = var **as** rel, if pathSyn(P2) ≠ NULL w(P1) = **ALL(rel in var** , if pathSyn(P2) ≠ NULL w(P1) = andSyn(P2) ret(P2).ret(A) w(O) ret(V) , if pathSyn(P2) == NULL ret(P1) = **count**(var) **> 0**, if pathSyn(P2) ≠ NULL ret(P1) = **count**(ret(P2)) **> 0**, if pathSyn(P2) == NULL

Säännössä 13 hahmo koostuu ainoastaan attribuutista, joka on kuvattu nonterminaalisymbolilla A. Jos hahmo esiintyy predikaatin ulkopuolella, se on tällöin XPath-kyselyn ainoa osa. Jos hahmo esiintyy predikaatissa, sillä voidaan testata jonkun attribuutin olemassaoloa predikaattiin sidotussa askelessa.

13. P → A

Säännön 13 semanttisissa säännöissä luodaan uusi muuttuja *var*, johon voidaan viitata myöhemmin kyselyssä. Tätä muuttujaa käytetään attribuutissa *q* sulkujen ympäröimänä. Näin kuvataan mitä tahansa solmua määrittelemättä kuitenkaan solmun nimeä. Tällöin lopullisen Cypher-kyselyn `MATCH`-lausekkeeseen tulee ainoastaan yksi solmu, jolla on tunnus muttei solmun tyyppiä. Synteettinen attribuutti *q* arvotetaan vain, kun sääntöä ei käytetä predikaatin sisällä, jotta Cypher-kyselyyn ei lisätä uutta solmua. Paluuarvoksi eli synteettisen attribuutin *ret* arvoksi arvotetaan uusi muuttuja *var*. Muuttujaa seuraa symbolilta A kuvautunut attribuutin *ret* arvo eli attribuutin nimi. Jos periytyneen attribuutin *insidePredicate* arvo on tosi eli sääntöä käytetään predikaatin sisällä, arvotetaan myös synteettinen attribuutti *w* `EXISTS`-funktiolla. Funktio testaa tällöin attribuutin olemassaoloa predikaattiin sidotusta solmusta tai kaaresta.

Synteettiset attribuutit

---

var = new_v()

*q(P) = **(**var**)** if insidePredicate(P) == FALSE*

*ret(P) = var**.**ret(A)*

*w(P) = **EXISTS**(appliesTo(P).ret(A)) , if insidePredicate(P)*

Säännöissä 14 ja 15 esitetään tilanne, kun polkuun kuuluu ainoastaan attribuutin arvon vertailu. Sääntöjä voidaan käyttää, kun XPath-kysely koostuu pelkästään vertailusta, tai vertailut ovat predikaatin sisällä.

14. P → A=V

15. P → A O V

Sääntöjen 11 ja 12 tapaan, säännöt 14 ja 15 palauttavat totuusarvon. Totuusarvo muodostetaan Cypher-kyselyssä säännöissä 11 ja 12 kuvatulla tavalla, joten sitä ei kerrata. Seuraavassa taulukossa on kuvattu säännön 15 semanttiset säännöt, jotka ovat lähes samanlaiset kuin säännöllä 14. Mikäli periytynyt attribuutti *appliesTo* on tyhjä, voidaan päätellä, että muodostettavaan Cypher-kyselyyn ei ole vielä liitetty osia. Tällöin arvotetaan kyselyn `MATCH`-osaan uusi kaarien ympäröimä Cypher-muuttuja kuvaamaan mitä

tahansa solmua graafitietokannassa. Jos periytynyt attribuutti *appliesTo* on arvotettu, voidaan päätellä, että sääntöä sovelletaan predikaatin sisällä. Tällöin synteettinen attribuutti *w* arvotetaan edellisten sääntöjen periaatteiden mukaisesti, ja vertailu sijoitetaan Cypher-kyselyn `WHERE`-lausekkeeseen.

Synteettiset attribuutit

---

var = new_v()

q(P) = **(** new_v() **)** , if appliesTo(P) == NULL

w(P) = andInh(P) var.ret(A) w(O) ret(V) , if appliesTo(P) == NULL

w(P) = andInh(P) appliesTo(P).ret(A) w(O) ret(V) , if appliesTo(P) ≠ NULL

ret(P) = **count(**var**) > 0**

andSyn(P) = **AND**

Säännössä 16 kuvataan edellisistä hahmosäännöistä hieman poikkeavaa sääntöä. Sääntö koostuu kahdesta nonterminaalisymbolista E ja P ja niitä erottavasta terminaalisymbolista /. Sääntö tarkoittaa, että kaarta voi seurata mikä tahansa hahmo, missä hahmon rakenne on joku säännöistä 6-15. Muihin hahmosääntöihin nähden symbolien järjestys on poikkeava, sillä nyt symboli E eli kaariaskel aloittaa hahmon. Poikkeuksellisuutta muista hahmoista korostaa symbolin merkintä PRP. Sääntöä 16 tulee käyttää XPath-ilmaisun predikaatin sisällä, jossa predikaatti on sidottu solmuaskeleeseen.

16. PRP → E **/** P

Semanttisissa säännöissä symbolin PRP synteettiset attribuutit *w* ja *andSyn* arvotetaan symboleilta E ja P kuvautuneilla vastaavilla synteettisillä attribuuteilla. Synteettinen attribuutti *q* arvotetaan symboleilta E ja P kuvautuneiden symbolien synteettisten attribuuttien *pq* arvoilla. Lisäksi attribuutin perään sijoitetaan symbolilta P kuvautuneen synteettisen attribuutin *q* arvo. Symbolilta P kuvautunut synteettinen attribuutti *q* kuvaa siis symbolin P XPath-ilmaisun mahdollisen predikaatin sisällä olevan semantiikan. Aliluvussa 4.2 predikaattia koskevan tulkinnan mukaan tämä merkkijono sijoitetaan Cypher-kyselyssä polkujen sekvenssin perään. Koska sääntöä 16 käytetään vain predikaatissa, ei synteettistä attribuuttia *ret* arvoteta. Synteettinen attribuutti *ret* kuvaa Cypher-kyselyn paluuarvoa, mutta XPathissa kyselyn paluuarvo ei voi esiintyä predikaatin sisällä. Periytyneistä attribuuteista *pathInh* ja *andInh* arvotetaan samaan tapaan kuin kuvattuna aiemmissa säännöissä 6-11. Lisäksi periytynyt attribuutti *insidePredicate* arvotetaan totuusarvolla tosi, koska sääntöä käytetään ainoastaan predikaatin sisällä.

Periytyneet attribuutit	*Synteettiset attribuutit*
pathInh(E) = pathInh(PRP)	q(PRP) = pq(E) pq(P) q(P)
andInh(E) = andInh(PRP)	w(PRP) = w(E) w(P)
andInh(P) = andSyn(E)	andSyn(PRP) = andSyn(P)
appliesTo(P) = ret(E)	
transitiveInh(P) = transitiveSyn(E)	
insidePredicate(E) = true	
insidePredicate(P) = true	

4.6.4 Solmuaskel (C)

Symboli C kuvaa XPath-ilmaisun solmuaskelta. Solmuaskel on XPathin graafitietokantojen semantiikassa kyselyyn kuuluva askel, joka kuvaa graafitietokannan solmua. Solmuaskel aloittaa XPath-ilmaisun, ja kaikki ilmaisuun kuuluvat polun. Solmuaskel esiintyy kaariaskelen jälkeen tai edellisen solmuaskelen jälkeen //-notaatiolla. Symbolille C voi soveltaa neljää sääntöä. Jokaisessa säännössä nonterminaalisymbolien järjestys on XPathin mukainen akseli, solmutesti ja viimeisenä predikaatti. Solmutestiä eli solmun nimeä kuvataan symbolilla N. Symboli N on ainoa symboli, joka kuuluu symbolin jokaiseen neljään sääntöön. Askeleeseen kuuluvaa akselia kuvataan lyhenteellä AX, ja nonterminaalisymboli predikaattia lyhenteellä PR. Akseli ja predikaatti ovat valinnaisia osia solmuaskelessa eivätkä esiinny kaikissa solmuaskeleen säännöissä. Kuten aliluvussa 4.2 selitettiin, akseli ei muuta solmuaskelen semantiikkaa. Tämän takia kaikissa solmuaskeleeseen sovellettavissa säännöissä nonterminaalisymboleilta AX ei kuvaudu synteettisiä attribuutteja. Symboli PR esiintyy aina työn terminaalisymboleihin kuuluvien hakasulkeiden sisällä.

Säännössä 17 esiintyy kaikki mahdollisesti solmuaskelen kuuluvat nonterminaalisymbolit. Tällöin säännössä määritellään askeleeseen kuuluva akseli, solmutesti ja predikaatti. Nonterminaalisymbolia seuraa terminaalisymboli ::, jolla XPathin kieliopissa erotetaan akseli solmutestistä.

17. C $\rightarrow$ AX::N[PR]

Semanttisissa säännöissä synteettisten attribuuttien *q* ja *w* arvot kuvautuvat symbolilta PR. Se on säännön ainoa symboli, jonka sisältämät rakenteet voivat sisältää suodattimen tai sisäkkäisen hahmon. Predikaattiin mahdollisesti kuuluva hahmo sijoitetaan tässä yhteydessä synteettiseen attribuuttiin *q*. Synteettiseen attribuuttiin *pq* taas sijoitetaan

symbolissa N arvotetun synteettisen attribuutin *q* arvo. Arvo sijoitetaan Cypher-kyselyssä sulkuparien sisälle. Sulkuparit kuvaavat Cypherissä solmua. Synteettisen attribuutin *pq* arvottaminen riippuu myös ehdosta synteettisen attribuutin *equal* suhteen. Jos attribuutti *equal* ei ole tyhjä, lisätään sulkuparin sisään symbolilta N kuvautuneen attribuutin *q* arvon lisäksi symbolilta PR synteettisen attribuutin *equal* arvo. Attribuutin *equal* arvo kuvautuu symbolilta PR, jos predikaatti sisältää yhtäsuuruusvertailun. Attribuutin *equal* arvo sijoitetaan aaltosulkeiden sisälle solmun yhteyteen. Cypherissä aaltosulkeilla voidaan solmun yhteydessä asettaa suodatin jonkin ominaisuuden yhtäsuuruuden testaamiseksi. Jos attribuutti *equal* on tyhjä, jätetään aaltosulkeet kirjoittamatta. Attribuutin *pq* arvottaminen riippuu myös periytyneen attribuutin *finishPath* arvosta. Jos periytynyt attribuutti *finishPath* on totuusarvoltaan tosi, lisätään attribuutin *pq* perään vielä pilkku ja kaarisulkeet. Kaarisulkeiden sisälle tulee symbolilta N kuvautunut synteettisen attribuutin *ret* arvo. Tämä sama arvo on solmun yhteyteen määritellyn muuttujan arvo Cypher-kyselyssä. Tämä toimenpide täytyy tehdä, jos solmuaskelta edeltänyt kaariaskel on sisältänyt transitiivisen akselin ja predikaatin. Aliluvussa 4.2 esiteltyjen periaatteiden mukaisesti tulee tämä hahmo erottaa omaksi osaksi Cypher-kyselyssä. Lisäksi synteettinen attribuutti *andSyn* arvotetaan symbolilta PR kuvautuneella vastaavalla attribuutilla.

Periytyneistä attribuutista *appliesTo* arvotetaan nonterminaalisymbolin N *ret* arvolla. Tätä attribuuttia käytetään välittämään tietoa predikaatille mihin muuttujaan se on sidottu. Koska yhtäsuuruusvertailu on aina solmu- tai kaarikohtainen, voidaan periytynyt attribuutti *dotInh* arvottaa tyhjäksi ennen synteettisten attribuuttien arvottamista. Symbolille PR välitettävä attribuutti *andInh* periytyy symbolilta C.

Periytyneet attribuutit	*Synteettiset attribuutit*
appliesTo(PR) = ret(N)	q(C) = q(PR)
dotInh(PR) = NULL	pq(C) = **(** q(N) **{**equal(PR)**})** , if equal(PR) ≠ NULL
andInh(PR) = andInh(C)	pq(C) = **(** q(N) **)** else
	pq(C) += **,(**ret(N)**)** , if finishPath(C)
	w(C) = w(PR)
	ret(C) = ret(N)
	andSyn(C) = andSyn(PR)

Säännössä 18 esiintyy nonterminaalisymbolit AX ja N. Lisäksi terminaalisymboli :: erottaa säännössä akselin ja solmutestin. Solmuaskeleeseen kuuluu siis vain askeleelle navigaatiosuunnan antava akseli ja solmutesti.

18. C → AX::N

Säännön 18 semanttisissa säännöissä synteettinen attribuutti *pq* arvotetaan nonterminaalisymbolilta N kuvautuneelta synteettisen attribuutin *q* arvolla, joka ympäröidään kaarisuluilla. Synteettinen attribuutti *pq* arvotetaan säännön 17 tavoin, mutta ilman synteettistä attribuuttia *equal* ja siihen liittyviä aaltosulkeita. Lisäksi synteettinen attribuutti *ret* arvotetaan tässäkin säännössä symbolilta N kuvautuneen vastaavan attribuutin arvolla. Yhtäkään periytynyttä attribuuttia ei arvoteta tässä säännössä.

Synteettiset attribuutit

pq(C) = **(**q(N)**)**
pq(C) += **,** **(**ret(N)**)** , if finishPath(C)
ret(C) = ret(N)

Säännössä 19 esiintyy nonterminaalisymbolit N ja PR. Symbolit kuvaavat solmutestiä ja predikaattia. Säännössä 20 esiintyy ainoastaan yksi nonterminaalisymboli N, joka kuvaa solmutestiä. Tällöin askelessa esiintyy siis vain solmun nimi. Koska akselilla ei ole solmuaskelessa semanttista merkitystä, semanttiset säännöt ovat säännöillä 19 ja 20 samat kuin säännöllä 17 ja 18, joten sääntöjä ei käydä läpi uudestaan.

19. C → N[PR]

20. C → N

4.6.5 Kaariaskel (E)

Symboli E kuvaa hahmoon kuuluvaa kaariaskelta. Kaariaskelessa kontekstivapaan kieliopin säännöt ovat samat kuin solmuaskelilla. Lisäyksenä sääntö 25, jossa käytetään säännön oikealla puolella ainoastaan terminaalisymbolia. Attribuuttikieliopissa solmu- ja kaariaskelten ero on semanttisissa säännöissä. Kaariaskelen semanttisissa säännöissä akselit vaikuttavat attribuuttien arvottamiseen.

Säännön 21 kontekstivapaa kielioppi vastaa solmuaskelen sääntöä 17.

21. E → AX::N[PR]

Kuten edellä selitettiin, sääntöjen 21 ja 17 eroavaisuus on semanttisissa säännöissä. Tästä osoituksena säännön 21 synteettisen attribuutin *pq* arvottaminen. Synteettiseen attribuuttiin *pq* lisätään tässä säännössä symbolilta AX kuvautuneet synteettiset attribuutit *start* ja *end*. Attribuutin *start* arvo sijoitetaan eteen ja attribuutin *end* loppuun. Näillä attribuuteilla kuvataan Cypher-kyselyyn liitettävää kaarta ja sen suuntaa. Attribuutti *pq* arvotetaan näillä synteettisillä attribuuteilla aina. Muuten attribuutin *pq* arvottaminen tapahtuu ehdoilla. Näitä ehtoja ovat synteettiset attribuutit *transitiveSyn* ja *equal*.

Synteettisen attribuutin *equal* mukaan arvottaminen käytiin läpi solmuaskelen yhteydessä säännössä 13. Myös tässä säännössä epätyhjä attribuutin *equal* arvo tarkoittaa aaltosulkeiden lisäämistä attribuuttiin *pq*. Synteettinen attribuutti *transitiveSyn* taas kuvautuu symbolilta AX ja sisältää totuusarvon kertoen onko symbolin AX kuvaama XPath-akseli transitiivinen. Näillä ehdoilla attribuutin *pq* arvottamista täydennetään seuraavasti. Jos attribuutti *transitiveSyn* on tosi, synteettinen attribuutti *pq* arvotetaan merkillä : ja nonterminaalisymbolilta N kuvautuneen synteettisen attribuutin *name* arvolla. Tämä johtuu siitä, että Cypherissä hahmossa kuvailtujen transitiivisten suhteiden yhteydessä ei voi käyttää muuttujaa, vaan ainoastaan kaaren nimeä kaksoispiste etuliitteenä. Jos kaaren akseli ei ole transitiivinen eli attribuutin *transitiveSyn* arvo on epätosi, sijoitetaan attribuuttiin *pq* symbolilta N kuvautuneen attribuutin *q* arvo. Muissa tapauksissa attribuutin *pq* arvottaminen tapahtuu suunnilleen samalla lailla kuin säännössä 17.

Säännössä arvotetaan synteettinen attribuutti *pathSyn*. Tämä attribuutti sisältää uniikin polun tunnuksen. Polun tunnusta tarvitaan myöhemmin kyselyssä, jos polkuun halutaan viitata kyselyssä myöhemmin. Attribuuttia *pathSyn* käytetään arvottamaan synteettinen attribuutti *pathSeq*. Aliluvussa 4.6.3 joissain hahmoissa oli mainittu synteettinen attribuutti *pathSeq* osana attribuuttia *pq*. Tämän attribuutin arvottaminen alkaa lisäämällä pilkku, joka erottaa tässä säännössä luodun uuden Cypherin polun edellisestä polusta. Tämän jälkeen synteettisen attribuutin *pathSyn* arvo lisätään osaksi attribuutin arvoa. Polun tunnusta seuraa yhtäsuuruusmerkki, jolla liitetään polun tunnus itse uuteen polkuun. Uusi polku aloitetaan periytyneen attribuutin *appliesTo* sisältämällä merkkijonolla. Tämä merkkijono sisältää kaariaskelta edeltäneeseen solmuun sidotun Cypherin muuttujan arvon. Koska Cypherissä polun pitää alkaa solmuaskelella, lisätään solmu Cypherin syntaksin mukaisesti polun alkuun. Muut synteettiset attribuutit *q*, *w*, *ret* ja *andSyn* arvotetaan suoraviivaisesti symboleilta E ja PR vastaavien attribuuttien arvoilla. Säännön 21 kaikki periytyneet attribuutit kuvautuvat nonterminaalisymbolille PR. Periytyneistä attribuuteista *appliesTo* arvotetaan kaariaskeleeseen kuuluvan symbolin N synteettisen attribuutin *ret* arvolla sitomaan XPath-ilmaisun predikaatti kaariaskeleeseen. Periytynyt attribuutti *dotInh* arvotetaan tyhjällä arvolla. Sen arvottaminen liittyy askeleeseen sidottuihin yhtäsuuruusvertailuihin. Näitä voi tulla askeleeseen useita, mutta sääntöön siirryttäessä niitä ei ole yhtään. Periytynyt attribuutti *andInh* arvotetaan tavalliseen tapaan symbolilta E kuvautuneen synteettisen attribuutin *andSyn* arvolla. Periytynyt attribuutti *appliesTo* arvotetaan säännössä solmutestisymbolin N synteettisen attribuutin *ret* arvolla. Myös

tieto akselin transitiivisuudesta välitetään predikaatille periytyneellä attribuutilla *transitiveInh*.

Periytyneet attribuutit	*Synteettiset attribuutit*
appliesTo(PR) = ret(N) dotInh(PR) = NULL transitiveInh(PR) = transitiveSyn(AX) pathInh(PR) = pathSyn(E) andInh(PR) = andInh(E) insidePredicate(PR) = true	pq(E) = start(AX):name(N) end(AX) if transitiveSyn(AX) == true and equal(PR) == NULL pq(E) = start(AX):name(N) { equal(PR) } end(AX) if transitiveSyn(AX) == true and equal(PR) ≠ NULL pq(E) = start(AX) q(N) { equal(PR) } end(AX) if transitiveSyn(AX) == false and equal(PR) ≠ NULL pq(E) = start(AX) q(N) end(AX) else q(E) = q(PR) w(E) = w(PR) ret(E) = ret(N) pathSyn(E) = new_v() pathSeq(E) =, pathSyn(E) = (appliesTo(E)) , if transitiveSyn(AX) == true andSyn(E) = andSyn(PR)

Sääntö 22 vastaa kontekstivapaassa kieliopissa sääntöä 18.

22. E → AX::N

Säännön 22 semanttisissa säännöissä arvotetaan ainoastaan synteettisiä attribuutteja. Synteettinen attribuutti *pq* arvotetaan samaan tapaan kuin säännössä 21. Myös synteettiset attribuutit *pathSyn* ja *pathSeq* arvotetaan samalla lailla ja samaa tarkoitusta varten kuin säännössä 21. Synteettinen attribuutti *ret* arvotetaan suoraan nonterminaalisymbolilta N kuvautuneella attribuutin arvolla. Toisin kuin säännössä 21, tässä säännössä ei arvoteta synteettisiä attribuutteja *q* tai *w*, koska kaariaskelella ei esiinny säännön 22 yhteydessä XPathin predikaattia. Säännössä vain predikaatti voisi sisältää hahmon tai Cypherin `WHERE`-lausekkeeseen sijoitettavaa attribuuttien vertailua.

Synteettiset attribuutit
pq(E) = start(AX) **:**name(N) end(AX) , if transitiveSyn(AX) = true
pq(E) = start(AX) q(N) end(AX) else
pathSyn(E) = new_v()
pathSeq(E) =**,** pathSyn(E) **= (**appliesTo(E) **)** , if transitiveSyn(AX) == true
ret(E) = ret(N)

Säännössä 23 säännön oikea puoli vastaa solmuaskelen säännön 15 oikeaa puolta.

23. E → N[PR]

Semanttiset säännöt ovat säännöllä 23 lähes samat kuin säännön 21 semanttisilla säännöillä. Periytyneet attribuutit arvotetaan samalla lailla. Säännössä 23 ei oteta huomioon ehtoa akselin transitiivisuuden suhteen, koska säännön oikea puoli ei sisällä akselia kuvaavaa symbolia AX. Kun XPathissa ei ole eksplisiittisesti määritetty akselia, se on oletuksena *child* eikä näin ollen ikinä transitiivinen. Taulukon 4 vastaavuuksien mukaan lisätään *child*-akselia vastaavat nuolet Cypher-kyselyyn. Koska säännössä esiintyy predikaatti, lisätään kaaren yhteyteen aaltosulkeet, jos synteettinen attribuutti *equal* on tosi. Muut synteettiset attribuutit arvotetaan vastaavien symbolilta PR kuvautuneiden synteettisten attribuuttien arvoilla.

Periytyneet attribuutit	*Synteettiset attribuutit*
appliesTo(PR) = ret(N)	q(E) = q(PR)
dotInh(PR) = NULL	pq(E) = **-[** q(N)
andInh(PR) = andInh(E)	pq(E) += **{** equal(PR) **}** , if equal(PR) ≠ NULL
insidePredicate(PR) = true	pq(E) += **]->**
	w(E) = w(PR)
	ret(E) = ret(N)
	andSyn(E) = andSyn(PR)

Säännössä 24 on säännön oikealla puolella vain solmutestiä kuvaava nonterminaalisymboli N. Näin ollen XPath-ilmaisussa kaaren määritelmänä on vain sen nimi.

24. E → N

Semanttiset säännöt säännölle 24 ovat yksinkertaiset. Akselia ei ole eksplisiittisesti määritelty säännössä, joten se tulkitaan akseliksi *child*. Säännöin 23 tavoin käytetään

lukusuuntaan oikealle suunnattua kaarta, jonka sisälle sijoitetaan nonterminaalisymbolilta synteettisen attribuutin *q* arvo. Säännön attribuutin *ret* arvoksi kuvautuu suoraan symbolin N attribuutin arvo.

Synteettiset attribuutit

pq(E) = **-[** q(N) **]->**
ret(E) = ret(N)

Säännössä 25 esiintyy ainoastaan terminaalisymboli. Tämä symboli kuvaa lyhennetyssä muodossa XPathin askelta, jossa akseli on *parent*, ja solmutesti villi kortti *. Tässä tapauksessa kaariaskelella otetaan huomioon kaikki solmuun saapuvat kaaret.

25. E → ..

Säännön 25 semanttiset säännöt muistuttavat säännön 24 semanttisia sääntöjä. Myöskään tässä säännössä periytyneitä attribuutteja ei arvoteta. Säännössä arvotetaan kaksi synteettistä attribuuttia. Koska sääntöön kuuluu akseli *parent*, lisätään sitä vastaavat nuolet Cypher-kyselyyn taulukon 4 vastaavuuksien mukaan. Nyt kaaren suunta on lukusuuntaan vasemmalle päin, koska *parent* navigoi kaaria käänteiseen suuntaan. Säännössä ei ole määritelty solmutestiä eli kaaren nimeä, joten luodaan uusi Cypher-muuttuja. Uusi muuttuja sijoitetaan nuolten väliin ja sillä arvotetaan myös synteettinen attribuutti *ret*. Uusi muuttuja täytyy luoda, jotta kaareen voi tarvittaessa viitata myöhemmin kyselyssä.

Synteettiset attribuutit

q(E) = **<-[** new_v() **]-**
ret(E) = new_v()

4.6.6 Attribuutti (A)

Symboli A kuvaa XPathin tietomallin attribuuttisolmua. XPathin graafitietosemantiikassa attribuuttisolmua käytetään hakemaan tietoa solmu- tai kaariaskeleeseen kuuluvasta attribuutista. Symboliin sovelletaan sääntöä 26. Säännössä terminaalisymbolina oleva merkkijono kuvaa attribuutin nimeä. Merkkijonoa edeltää attribuuttia merkitsevä terminaalisymboli @.

26. A → *@a_name*

Semanttisissa säännöissä tuotetaan arvo ainoastaan synteettiselle attribuutille *ret*, joka kuvaa Cypher-kyselyn paluuarvoa. Säännössä 26 attribuutti *ret* arvotetaan XPathilmaisun terminaalisymbolina olevalla attribuutin nimeä kuvaavalla merkkijonolla.

Synteettiset attribuutit
ret(A) = a_name

4.6.7 Solmutesti (N)

Solmutesti on XPath-ilmaisuun kuuluvissa askelissa pakollinen osa. Solmutestiä käytetään tässä työssä kuvaamaan solmu- tai kaariaskelen nimeä. Solmutesti lyhennetään kirjaimella N. Tähän symboliin voidaan soveltaa kolmea sääntöä. Jokaisessa kolmessa säännössä arvotetaan ainoastaan synteettisiä attribuutteja.

Säännössä 27 säännön oikea puoli sisältää vain terminaalisymbolina toimivan merkkijonon. Tämä merkkijono kuvaa XPathin tietomallin solmun nimeä.

27. N → *name*

Synteettisissä attribuuteissa luodaan uusi uniikki muuttuja, jotta tähän voidaan viitata myöhemmin Cypher-kyselyn eri vaiheissa. Sama muuttujan nimi annetaan arvoksi myös Cypher-kyselyn paluuarvon muodostavaan synteettisen attribuuttiin *ret*. Muuttuja esitellään Cypher-kyselyn `MATCH`-lausekkeessa, joten muuttujan nimi sijoitetaan osaksi synteettisen attribuutin *q* arvoa, joka lopulta tulee osaksi `MATCH`-lauseketta. Attribuuttiin *q* sijoitetaan myös XPath-ilmaisussa annettu solmutestin nimi kaksoispisteellä eroteltuna. Solmutestin nimellä arvotetaan myös synteettinen attribuutti *name*.

Synteettiset attribuutit
q(N) = new_v() **:** *name*
name(N) = *name*
ret(N) = new_v()

Säännössä 28 esiintyy vain yksi terminaalisymboli säännön oikealla puolella. Tämä terminaalisymboli kuvaa villiä korttia, jolla ei rajoiteta askelen tuloksia solmutestin arvon perusteella. XPathissa sääntö 28 viittaa oletuksena siirtymisenä vanhemmasta lapseen, koska akselia ei ole askelessa eksplisiittisesti määritelty.

28. N → *

Semanttisissa säännöissä luodaan uusi muuttuja, jolla arvotetaan synteettiset attribuutit *q* ja *ret*. Cypherissä ei ole eksplisiittistä merkintätapaa villille kortille, vaan sitä kuvataan sijoittamalla muuttujan nimi Cypherin elementin eli solmun tai kaaren sisälle.

Synteettiset attribuutit
q(N) = ret(N) = new_v()

Säännössä 29 esiintyy terminaalisymbolina kaksi pistettä, jotka kuvaavat XPathissa villiä korttia. Siinä missä säännössä 28 villi kortti * kuvasi siirtymistä vanhemmasta lapseen, terminaalisymboli .. kuvaa siirtymistä lapsesta sen vanhempaan.

29. N → ..

Koska symboli N kuuluu sekä solmu-, että kaariaskeleeseen, voi sääntöä soveltaa kummassa vain. Koska kaariaskelella oli säännössä 25 vastaava säännön oikea puoli tarkoittaen samaa asiaa, käytännössä tätä sääntöä sovelletaan vain solmuaskelen yhteydessä. Koska solmuaskelessa ei oteta huomioon askelen sisältämää akselia, sääntö on semanttisesti redundantti. Säännön semanttiset säännöt ovat samat kuin säännössä 28.

4.6.8 Akseli (AX)

XPathissa akseli kuvaa navigaatiosuuntaa edellisestä solmusta. Symbolia akseli merkitään tämän työn kontekstivapaassa kieliopissa AX. Symboliin voi soveltaa kuutta erilaista sääntöä, joista jokainen vastaa yhtä työhön valituista XPathin akseleista. XPathin akselit esitetään työssä sellaisena, kuin ne on ilmaistu virallisessa XPathin dokumentaatiossa. Jokaisen akseliin liittyvän säännön oikea puoli sisältää yhden terminaalisymbolin. Terminaalisymboli on merkkijono, jonka arvo vaihtelee akselin mukaan. Tämän arvon perusteella jokaisen symbolin AX semanttisissa säännöissä arvotetaan synteettiset attribuutit *start* ja *end*. Attribuuttien arvottaminen tapahtuu taulukon 4 vastaavuuksien mukaan. Attribuutti *start* säilöö kaaren alun ja attribuutti *end* lopun. Cypherin syntaksissa kaaren sisälle sijoitetaan kaaren nimi, muuttuja sekä mahdolliset yhtäsuuruustestit. Nämä tiedot lisätään kaaren osien väliin kaariaskelen semanttisissa säännöissä. Lisäksi synteettinen attribuutti *transitiveSyn* välittää tiedon onko symbolin AX sisältämä akseli transitiivinen. Synteettinen attribuutti *transitiveSyn* arvotetaan todeksi säännöissä 44, 45, 46 ja 47. Säännöissä 43 ja 44 navigointi etenee vain vierussolmuun, jolloin akseli ei ole transitiivinen.

Säännössä 42 tarkasteltava akseli on *child*. XPathin graafitietokantasemantiikassa tämä akseli kuvaa siirtymistä kaaren suuntaa pitkin yhden askelen verran. *Child* on oletusakseli jokaisessa askelessa, joten säännön 42 käyttö lienee harvinaista.

42	AX → child	start(AX) = **-[** end(AX) = **]->** transitiveSyn(AX) = false

Säännössä 43 nonterminaalin arvo on *parent*. XPathin graafitietokannan semantiikassa tämä tarkoittaa siirtymistä solmusta vierussolmuun kulkien näiden välistä kaarta käänteiseen suuntaan.

43	AX → parent	start(AX) = **<-[** end(AX) = **]-** transitiveSyn(AX) = false

Sääntö 44 sisältää merkkijonon *descendant* terminaalisymbolina. XPathissa *descendant* hakee kontekstisolmun jälkeläissolmut. Graafitietokannan semantiikassa akselilla haetaan mielivaltainen polku kahden solmun välillä. Jos polun päätesolmua ei ole määritelty, haetaan kaikki polut lähtien alkusolmusta. Kaikkia polkuun kuuluvia kaaria pitkin edetään aina niiden suuntaan. Semanttisissa säännöissä synteettiset attribuutit *start* ja *end* arvotetaan jälleen taulukon 4 vastaavuuksien mukaan. Akseli on transitiivinen. Tätä ominaisuutta kuvaa Cypherin syntaksissa asteriski.

44	AX → descendant	start(AX) = **-[** end(AX) = ***]->** transitiveSyn(AX) = true

Sääntö 45 sisältää merkkijonon *ancestor* terminaalisymbolina. XPathissa *ancestor* hakee kontekstisolmun esivanhempi-solmuja. Graafitietokannan semantiikassa tällä akselilla haetaan mielivaltaisen pituisia polkuja kahden solmun välillä. Polut koostuvat kaarista, joita pitkin kuljetaan aina käänteiseen suuntaan. Kuten säännössä 44, tässäkin transitiivisuutta kuvaava asteriski lisätään osaksi Cypher-ilmaisun kaarta.

45	AX → ancestor	start(AX) = **<-[** end(AX) = ***]-** transitiveSyn(AX) = true

Säännössä 46 nonterminaalisymbolina olevan merkkijonon arvo on *descendant-or-self*. Akseli täydentää akselia *descendant* refleksiivisyyttä ilmaisevalla liitteellä *self*. Tällä akselilla haetaan samat solmut kuin säännössä 44, mutta lisäksi tyhjä polku, jolla palautetaan myös solmu, josta navigointi alkoi. Semanttisissa säännöissä refleksiivisyys-transitiivisuutta kuvataan Cypherissä asettamalla asteriskin jälkeen lukumäärä kaarten minimi- ja maksimilukumäärälle. Lukumäärät voivat olla positiivisia kokonaislukuja. XPathin semantiikassa akselilla palautetaan tyhjä polku ja muut polut ottamatta kantaa polun maksimipituuteen. Tällöin sijoitetaan kaaren yhteyteen Cypher-kyselyyn merkintä `0..`, missä 0 tarkoittaa tyhjää polkua ja kaksoispiste rajoittamatonta polun pituutta.

46	AX → descendant-or-self	start(AX) = **-[** end(AX) = ***0..]->** transitiveSyn(AX) = true

Säännössä 47 terminaalisymbolin arvona on *ancestor-or-self*, joka lisää niin ikään refleksiivisyyden akselille *ancestor* samaan tapaan kuin säännössä 46 *descendant*-akselin suhteen. Samaan tapaan kuin säännössä 46, Cypher-ilmaisussa kaareen asetetaan refleksiivisyys-transitiivisuudesta kertova polun minimi- ja maksimipituus.

47	AX → ancestor-or-self	start(AX) = **<-[** end(AX) = ***0..]->** transitiveSyn(AX) = true

4.6.9 Predikaatti (PR)

Predikaatti on osa XPath-ilmaisuun kuuluvaa askelta, jolla asetetaan akselin ja solmutestin lisäksi ehdot askelelle. Predikaatit sallivat monipuolisempien ehtojen asettamisen, ja niillä pystytään mallintamaan XPathissa polkujen haarautumista. Predikaattia kuvataan tässä työssä symbolilla PR. Symboliin PR voidaan soveltaa kuutta eri sääntöä. Erona muihin symboleihin, predikaatin sääntöjen tuottamien merkkijonojen paikka Cypher-kyselyssä riippuu pitkälti kunkin säännön semanttisista säännöistä itsessään. Erona myös moneen muuhun symboliin tässä työssä, predikaatit eivät voi sisältää kyselyn projektiota. Tämän takia symbolin PR säännöissä ei arvoteta Cypher-kyselyn paluuarvon määrittävää synteettistä attribuuttia *ret*. Säännöissä 37-41 esitetään millaisia ilmaisuja predikaatin sisällä voi tämän työn rajauksessa esiintyä. Säännössä 36 esitetään usean predikaatti-ilmaisun ketjuttamista loogisella konjunktiolla.

Säännössä 37 predikaatti on määritelty yhdellä nonterminaalisymbolilla PRP. Symboli PRP oli kuvattu yksityiskohtaisemmin aliluvun 4.6.3 säännössä 16. Predikaattisymbolin voi säännössä 37 siis muodostaa vain hahmoa kuvaava nonterminaalisymboli.

37. PR → PRP

XPathin predikaatti vastaa suodatinta ja Cypherissä sitä vastaa yleensä lauseke `WHERE`. Cypherin `WHERE`-lauseke kuitenkin rajoittaa tässä tapauksessa ilmaisuvoimaa, koska lausekkeessa ei voi esitellä uusia muuttujia. Säännössä 37 XPath-ilmaisun predikaatti voi kuitenkin sisältää sisäkkäisiä predikaatteja mahdollistaen haarautuvien hahmorakenteiden ilmaisun. Tämä vaatisi Cypher-kyselyssä nimenomaan uusien muuttujien esittelyn, että voidaan mallintaa mahdolliset haarautuvien hahmorakenteet. Koska XPathissa predikaattiin sidottu muuttuja kuuluu aiemmin esiteltyyn hahmoon, uuden hahmon

yhteys muihin ilmaisun hahmoihin täytyy pystyä mallintamaan. Tämän takia säännön 37 predikaatin sisältävä hahmo olisi haastava mallintaa Cypher-ilmaisun `WHERE`-lausekkeeseen. Parempana vaihtoehtona hahmo sijoitetaan Cypher-ilmaisussa `MATCH`-lausekkeeseen. `MATCH`-lauseke sisältää polkujen sekvenssin, joista jokaiselle polulle täytyy löytyä graafitietokannassa täsmäys, että kysely evaluoidaan todeksi. Aliluvun 4.2 idean mukaan XPath-ilmaisu pilkotaan predikaattien kohdalla poluiksi, missä aiemman polun loppusolmu on predikaatin edessä oleva solmu. Samalla tämä solmu on uuden polun alkusolmu. Näiden polkujen semantiikkaa vastaavat Cypher-ilmaisut sijoitetaan sitten `MATCH`-lausekkeeseen muodostaen XPathin predikaatteja vastaavan semantiikan. Tämän takia säännössä 37 arvotetaan synteettinen attribuutti *q*, joka lopulta sijoitetaan Cypher-kyselyn `MATCH`-lausekkeeseen. Tämän attribuutin eteen sijoitetaan säännössä pilkku erottamaan predikaattia edeltäneen polun symbolin PRP sisältämästä polusta. Yhteys edelliseen polkuun luodaan periytyneellä attribuutilla *appliesTo*, joka sisältää predikaattiin sidotun muuttujan nimen. Attribuutin *appliesTo* arvo sijoitetaan sulkujen sisään pilkun jälkeen. Kuten aliluvussa 4.6.3 ja sen säännössä 16 todettiin, symbolissa PRP esiintyy ensimmäisenä aina kaariaskel. Koska kaariaskelta edeltää solmuaskel, attribuutin *appliesTo* arvo oletetaan solmuaskeleeseen liittyväksi muuttujaksi, jonka takia kaarisulkuja käytetään. Lisäksi symbolin PRP tuottama attribuutin *q* arvo lisätään osaksi symbolin PR attribuutin *q* arvoa. Muut synteettiset attribuutit *w* ja *andSyn* kuvautuvat sellaisenaan symbolilta PRP. Periytyneitä attribuutteja esiintyy säännössä yksi. Tämä attribuutti on *andInh*, jonka arvottamisperiaatteita on käyty aiemmissa säännöissä läpi.

Periytyneet attribuutit	*Synteettiset attribuutit*
andInh(PRP) = andInh(PR)	q(PR) = , (appliesTo(PR)) q(PRP) w(PR) = w(PRP) andSyn(PR) = andSyn(PRP) equal(PR) = NULL

Sääntö 38 kuvaa tapausta, jossa predikaattiin on sijoitettu arvon vertailu. Säännön 38 oikea puoli sisältää kolme nonterminaalisymbolia. Symboli A kuvaa XPath-ilmaisun attribuuttia, symboli O operaattoria ja symboli V arvoa. Arvo voi olla tässä tapauksessa olla merkkijono, numeerinen arvo tai totuusarvo.

38. PR $\rightarrow$ A O V

Semanttisissa säännöissä arvotetaan ainoastaan synteettisiä attribuutteja. Toisin kuin säännössä 37, säännön 38 tuottama Cypher-ilmaisu sijoitetaan Cypher-kyselyssä aina `WHERE`-lausekkeeseen. Tämän takia Cypher-ilmaisu sijoitetaan synteettiseen attribuuttiin *w* arvo. Attribuutin *w* alkuun sijoitetaan periytyneen attribuutin *andInh* arvo. Attribuutti *andInh* sisältää joko arvon AND tai tyhjän arvon riippuen siitä, onko käännöksessä aiemmin lisätty synteettiseen attribuuttiin *w* sisältöä. Tämän jälkeen synteettisen attribuutin *w* arvottaminen riippuu periytyneen attribuutin *transitiveInh* sisältämästä totuusarvosta. Attribuutti *transitiveInh* kertoo tässäkin säännössä, mikäli predikaattiin sidotulla kaariaskelella on transitiivinen akseli. Jos *transitiveInh* on tosi, attribuutti *w* arvotetaan lisäämällä attribuuttiin relationships-funktio. Funktiolla annetaan parametriksi periytyneen attribuutin *pathInh* arvon, joka kuvaa polun tunnusta. Funktiolla saadaan muodostettua lista, joka sisältää kaikki polkuun kuuluvat suhteet. Niitä iteroidaan läpi käyttäen varattua avainsanaa `in`, ja luomalla uusi Cypher-muuttuja. Muuttujaa käytetään viittaamaan iteroidessa listaan kuuluviin suhteisiin. Lauseke sijoitetaan `ALL`-funktion sisälle, joka on Cypherissä predikaattifunktio. Funktiolla suodatetaan pois kaikki sellaiset polut, joiden suhteet eivät täytä funktiossa asetettua ehtoa. Ehto asetetaan `WHERE`-lausekkeella, jonka näkyvyys on rajattu tässä tapauksessa ainoastaan `ALL`-funktion sisälle. Jos attribuutti *transitiveInh* on epätosi, lisätään attribuuttiin *w* ainoastaan periytyneen attribuutin *appliesTo* arvo. Riippumatta attribuutin *transitiveInh* arvosta, synteettiseen attribuuttiin *w* lisätään piste, attribuutin nimi, Cypher-operaattori ja verrattava arvo. Lopussa suljetaan predikaattifunktio `ALL` kaarisululla, jos siis attribuutti *transitiveInh* on tosi. Lisäksi synteettinen attribuutti *andSyn* arvotetaan arvolla AND, koska nyt Cypher-kyselyn `WHERE`-lausekkeeseen on lisätty sisältöä. Seuraavalla kerralla kun lisätään sisältöä synteettiseen attribuuttiin *w*, ketjutetaan uusi ehto Cypherin AND-avainsanalla.

Synteettiset attribuutit

---

w(PR) = andInh(PR)

w(PR) += **ALL(rel in relationships(** pathInh(PR) **) WHERE** , if transitiveInh(PR)

w(PR) += **rel** , if transitiveInh(PR)

w(PR) += appliesTo(PR) else

w(PR) += **.** ret(A) w(O) q(V)

w(PR) += **)** , if transitiveInh(PR)

andSyn(PR) = **AND**

Säännössä 39 verrataan attribuuttia ja arvoa yhtäsuuruudella. Säännössä 39 ainoa poikkeus sääntöön 38 on, että symbolin O sijaan säännön oikealla puolella esiintyy non-terminaalisymbolin O sijaan terminaalisymboli =.

39. PR → A = V

Säännössä 39 esiintyvä yhtäsuuruusvertailu sijoitetaan Cypher-kyselyssä `MATCH`-lausekkeeseen. Vertailu sijoitetaan Cypherin syntaksissa solmun tai kaaren yhteyteen riippuen siitä kumpaan XPath-ilmaisun predikaatti on sidottu. Attribuutin *equal* arvo sijoitetaan Cypher-kyselyssä solmu- tai kaariaskelen yhteyteen aaltosulkeiden sisälle. `MATCH`-lausekkeen lisäksi vertailun voi myös sijoittaa `WHERE`-lausekkeeseen. Sijoitus osana solmua tai kaarta on kuitenkin luontevampaa ja vähentää viittauksia muuttujiin Cypher-kyselyn sisällä. Säännössä arvotetaan ainoastaan synteettiset attribuutit *equal* ja *dotSyn*. Solmuun tai kaareen voi kiinnittää useamman yhtäsuuruusvertailun, ja nämä täytyy erottaa toisistaan pilkulla. Tällöin täytyy tietää, onko samaan solmuun tai kaareen jo lisätty yhtäsuuruusvertailu. Tämä onnistuu periytyneellä attribuutilla *dotInh*. Attribuutti sisältää joko pilkun tai tyhjän arvon riippuen onko solmussa tai kaaressa olemassa jo yhtäsuuruusvertailu. Tämän jälkeen synteettiseen attribuuttiin *equal* lisätään attribuutin nimi, joka kuvautuu symbolilta A synteettisellä attribuutilla *q*. `MATCH`-lausekkeessa yhtäsuuruusoperaattoria vastaa kaksoispiste. Operaattorin jälkeen attribuutille lisätään vielä symbolilta V kuvautuneen merkkijonon arvo. Säännössä 39 arvotetaan synteettinen attribuutti *dotSyn*. Arvoksi tämä attribuutti saa pilkun. Attribuutin *dotSyn* arvo sijoitetaan jatkossa uuden yhtäsuuruusvertailun eteen, jos sellainen lisätään samaan solmuun tai kaareen Cypher-kyselyssä.

Synteettiset attribuutit
*equal(PR) = dotInh(PR) q(A) **:** q(V)*
dotSyn(PR) = **,**

Säännön 40 oikealla puolella esiintyy vain attribuuttia kuvaava symboli A. Tällä säännöllä tarkastellaan XPathissa, onko predikaattiin sidotussa askelessa olemassa symbolin A kuvaamaa attribuuttia. XPathin semantiikassa graafitietokannoissa säännössä tarkastellaan sisältääkö predikaattiin sidottu solmu tai kaari yhtenä ominaisuutena symbolin A kuvaamaa ominaisuutta.

40. PR → A

Säännössä 40 arvotetaan ainoastaan synteettisiä attribuutteja. Säännön 40 semanttisissa säännöissä esiintyy useita yhtäläisyyksiä sääntöön 38. Tässäkin säännössä synteettisen attribuutin arvottaminen riippuu periytyneen attribuutin *transitiveInh* arvosta. Cypherin funktiota `relationships` käytetään, jos attribuutti *transitiveInh* on tosi. Ehdon ja Cypherin funktion toimintaperiaate jätetään kertaamatta. Ominaisuuden olemassaoloa testataan Cypherissä predikaattifunktiolla `EXISTS`. Funktion toimintaperiaate kuvailtiin aliluvun 4.6.3 säännössä 10.

Synteettiset attribuutit
w(PR) = andInh(PR)
w(PR) += **ALL(rel in relationships(** pathInh(PR) **) WHERE** , if transitiveInh(PR)
w(PR) += **EXISTS(rel .** ret(A)**)** , if transitiveInh(PR)
w(PR) += **EXISTS(** appliesTo(PR) **.** ret(A)**)** else
w(PR) += **)** , if transitiveInh(PR) == true
andSyn(PR) = **AND**

Säännössä 41 predikaattisymboli PR sisältää nonterminaalisymbolin SF. Symboli SF kuvaa merkkijonofunktiota. Työssä käytetyt merkkijonofunktiot on tarkemmin kuvattu aliluvussa 4.6.10.

41: PR → SF

Säännön 41 semanttisissa säännöissä arvotetaan kaksi periytynyttä ja synteettistä attribuuttia. Periytyneiden attribuuttien *transitiveInh* ja *appliesTo* arvot välitetään muuttumattomana edelleen symbolille SF välittämään tietoa predikaattiin sidotun XPath-askelen tiedoista. Symbolilta SF kuvautunut synteettisen attribuutin *w* arvo liitetään osaksi symbolin PR vastaavan attribuutin arvoa. Attribuutin *w* eteen sijoitetaan myös periytyneen attribuutin *andInh* arvo ketjuttamaan muita mahdollisia ehtoja. Samasta syystä kuin säännöissä 38 ja 40, myös tässä säännössä synteettinen attribuutti *andSyn* arvotetaan merkkijonolla AND.

Periytyneet attribuutit	*Synteettiset attribuutit*
transitiveInh(SF) = transitiveInh(PR)	w(PR) = andInh(PR) w(SF)
appliesTo(SF) = appliesTo(PR*)*	andSyn(PR) = **AND**

Kuten aliluvun 4.6.4 ja 4.6.5 säännöissä näytettiin, yhteen XPathin askeleeseen kuuluu korkeintaan yksi predikaatti. Tämän rajoituksen voi kuitenkin XPathissa kiertää

ketjuttamalla predikaatteja yhden predikaatin sisällä erilaisilla operaattoreilla. Tässä työssä havainnollistetaan predikaattien ketjuttamista loogisella konjunktiolla, jota kuvataan säännössä 36. Säännössä 36 predikaattiin kuuluu kaksi nonterminaalisymbolia PR2 ja PR3. Symbolit on ketjutettu terminaalisymbolilla *and*, joka kuvaa XPathissa loogista konjunktiota. Nonterminaalisymbolit PR2 ja PR3 kuvaavat predikaatteja. Koska XPathin predikaatti toimii totuusarvon palauttavana funktiona, XPathissa evaluoidaan säännön 36 tapauksessa predikaatin PR1 paluuarvoksi tosi ainoastaan, kun molemmat predikaatit palauttavat totuusarvon tosi. XPathin semantiikassa graafitietokannassa tämä pätee, kun molemmille säännössä kuvailluille predikaateille on olemassa täsmäys graafitietokannassa.

36. PR1 → PR2 and PR3

Vaikka Cypherin syntaksissa on olemassa operaattori `AND` loogiselle konjunktiolle, se ei aivan vastaa XPathin *and*-operaattorin ideaa. XPathissa *and*-operaattorilla voidaan ketjuttaa predikaatin sisällä ehtoja poluista, hahmoista, attribuuteista tai suuruusvertailuista. Tämä on suora seuraus XPathin syntaksista, jossa kyselyn ominaisuudet, kuten hahmon täsmäys, paluuarvo ja suodattimet esiintyvät ilmaisussa sekoittuneena. Cypherissä operaattorin `AND` käyttäminen on rajoitetumpaa. Operaattorilla `AND` voidaan kyllä Cypherissä ketjuttaa toisiinsa samoja ominaisuuksia kuin XPathissa, mutta Cypherissä operaattoria `AND` voi esiintyä vain kyselyn `WHERE`-lausekkeessa. Kuten säännössä 37 näytettiin, XPathin predikaatin sisällä olevat hahmot mallinnetaan tässä työssä kyselyn `MATCH`-lausekkeeseen. Tällaisten predikaattien yhteydessä operaattoria `AND` ei lisätä Cypher-kyselyyn. Näissä tapauksissa XPath-ilmaisun konjunktio on tulkittavissa Cypher-kyselyn `MATCH`-lausekkeen rakenteesta. `MATCH`-lausekkeessa kuvataan useita polkuja, joista jokaiselle täytyy löytyä täsmäys, jotta Cypher-kysely evaluoidaan todeksi. Sen sijaan säännöissä 38-41 predikaattien yhteydessä Cypher-kyselyyn lisätään operaattori `AND` erottamaan `WHERE`-lausekkeeseen sijoitettavia ehtoja toisistaan.

Semanttisissa säännöissä pääasiassa kaikki synteettiset attribuutit arvotetaan kahden nonterminaalisymbolin PR2 ja PR3 tuottamien synteettisten attribuuttien arvoilla. Synteettisen attribuutin *q* arvot kuvautuvat symbolien PR2 ja PR3 vastaavalta attribuutilta, ja ne asetetaan peräkkäin tässä järjestyksessä. Nämä arvot kuvaavat mahdollisesti Cypher-kyselyn `MATCH`-lausekkeeseen sijoitettavia kyselyn osia. `MATCH`-lausekkeeseen sijoitetaan lopulta myös synteettisen attribuutin *equal* arvo. Arvottaminen tapah-

tuu symboleilta PR2 ja PR3 kuvautuneiden vastaavien attribuutin arvojen konkatenaatiolla. Samalla lailla arvotetaan säännössä 36 myös synteettinen attribuutti *w*. Lisäksi pilkun sijoittamisesta yhtäsuuruusvertailun sisään kertova synteettinen attribuutti *dotSyn* arvotetaan symbolin PR3 arvottamalla arvolla. Myös synteettinen attribuutti *andSyn* arvotetaan symbolilta PR3 kuvautuneen vastaavan attribuutin arvolla. Periytyneiden attribuuttien *transitiveInh*, *pathInh*, *appliesTo* arvot ovat samat symboleille PR2 ja PR3, koska molemmat alipredikaatit on sidottu samaan askeleeseen XPath-kyselyssä. Muiden periytyneiden attribuuttien *andInh* ja *dotInh* arvot voivat muuttua alipredikaateissa. Tämän takia symboli PR2 saa kummankin periytyneen attribuutin arvoksi aina symbolilta PR1 periytyneen arvon. Vastaavasti symboli PR3 perii synteettisten attribuuttien *dotInh* ja *dotSyn* arvot symbolilta PR2.

Periytyneet attribuutit	*Synteettiset attribuutit*
andInh(PR2) = andInh(PR1)	q(PR1) = q(PR2) q(PR3)
appliesTo(PR2) = appliesTo(PR1)	w(PR1) = w(PR2) w(PR3)
dotInh(PR2) = dotInh(PR1)	equal(PR1) = equal(PR2) equal(PR3)
pathInh(PR2) = pathInh(PR1)	andSyn(PR1) = andSyn(PR3)
transitiveInh(PR2) = transitiveInh(PR1)	dotSyn(PR1) = dotSyn(PR3)
andInh(PR3) = andSyn(PR2)	
appliesTo(PR3) = appliesTo(PR1)	
dotInh(PR3) = dotSyn(PR2)	
pathInh(PR3) = pathInh(PR1)	
transitiveInh(PR3) = transitiveInh(PR1)	

4.6.10 Merkkijonofunktiot (SF)

Merkkijonofunktiot ovat työssä suodattimina toimivia funktioita, joita voi käyttää XPath-ilmaisussa predikaatin sisällä. Merkkijonofunktioita kuvataan työssä symbolilla SF. Merkkijonofunktioon käytetään työssä yhtä sääntöä. Tämä sääntö koostuu kahdesta nonterminaalisymbolista, ja neljästä terminaalisymbolista. Nonterminaalisymboli SFN kuvaa funktion nimeä. Tämä symboli ja mahdolliset työhön valittujen funktioiden nimet esitellään taulukossa 8. Funktiolla on terminaalisymboleina kaarisulut kuvaamassa parametrilistaa. Työhön valitut merkkijonofunktiot ottavat aina kaksi parametria. Ensimmäinen parametri on attribuuttisymboli A, joka esitettiin aliluvussa 4.6.6. Toinen parametri on

merkkijonoa kuvaava terminaalisymboli. Terminaalisymbolina toimiva pilkku erottaa parametrit. Funktiossa verrataan siis attribuutin arvoa toisena parametrina esiteltyyn merkkijonoon. Jos merkkijono täsmää attribuutin arvoon, funktio evaluoituu todeksi.

48. SF → SFN(A, *"string"*)

Säännön 48 semanttisissa säännöissä käytetään useita samoja käytäntöjä kuin aliluvun 4.6.9 säännöissä 38 ja 40. Näitä samoja käytäntöjä ei kerrata tässä yhteydessä. Säännössä synteettiseen attribuuttiin *w* lisätään symbolilta A kuvautuneen attribuutin *ret* arvo, jota seuraa symbolilta SFN kuvautunut synteettisen attribuutin *w* arvo. Tämän attribuutin arvona on XPath-ilmaisun merkkijonofunktion vastine Cypherissä. Tämän jälkeen attribuuttiin *w* lisätään XPath-ilmaisussa funktion toisen parametrin merkkijono. Tämä merkkijono ympäröidään Cypherissä yksinkertaisilla lainausmerkeillä.

Synteettiset attribuutit

w(SF) += **ALL(rel in relationships(** pathInh(SF) **) WHERE** , if transitiveInh(SF)
w(SF) += **rel** , if transitiveInh(SF) == true
w(SF) += appliesTo(SF) else
w(SF) += **.** ret(A) w(SFN) **'***string***'**
w(SF) += **)** , if transitiveInh(SF) == true

Merkkijonofunktion nimeen viitataan symbolilla SFN. Merkkijonofunktion nimiin voidaan soveltaa tässä työssä kolmea eri sääntöä, joista kukin vastaa yhtä funktion nimeä. Työhön valitut XPathin merkkijonofunktiot ovat *contains*, *starts-with* ja *ends-with*. Funktio *contains* testaa löytyykö täsmättävä merkkijono attribuutin sisältämästä merkkijonosta. Funktio *starts-with* testaa alkaako merkkijono funktion toisen parametrin mukaisella merkkijonolla. Funktio *ends-with* tarkistaa loppuko merkkijono funktion toisen parametrin merkkijonolla. Valituilla funktioilla havainnollistetaan, millaisia ehtoja merkkijonoille voidaan asettaa XPathin graafitietokantasemantiikassa. Jokaiselle funktion nimistä löytyy suoraviivainen vastinpari Cypheristä. Cypherissä edellä lueteltuja XPathin funktion nimiä vastaavat ilmaisut `CONTAINS`, `STARTS WITH` ja `ENDS WITH`. Merkkijonofunktioiden nimien semanttisten sääntöjen tuottama käännös liitetään Cypher-kyselyn `WHERE`-lausekkeeseen. Tällöin säännöissä arvotetaan synteettinen attribuutti *w*. Koska jokaisen säännön semanttiset säännöt ovat suhteellisen yksinkertaiset, niin sanallisen selityksen sijaan semanttiset säännöt 49-51 on esitetty taulukossa 8.

49	SFN → contains	w(SFN) = `CONTAINS`
50	SFN → starts-with	w(SFN) = `STARTS WITH`
51	SFN → ends-with	w(SFN) = `ENDS WITH`

Taulukko 8. Symbolin SFN kontekstivapaa kielioppi ja semanttiset säännöt

4.6.11 Operaattorit (O)

XPathiin kuuluu erilaisia operaattoreita. Operaattoria kuvataan tässä työssä symbolilla O. XPathin operaattoreista työssä tarkastellaan vertailuoperaattoreita, jotka tarkastelevat arvojen yhtäsuuruutta tai erisuuruutta. Operaattorit sijoitetaan tavallisesti Cypher-kyselyn `WHERE`-lausekkeeseen. Poikkeuksena on luvun 4.6.9 säännön 39 yhtäsuuruusoperaattori, joka sijoitetaan `MATCH`-lausekkeeseen. Koska yhtäsuuruusoperaattorille on oma sääntönsä predikaattien yhteydessä, sääntöä 30 tuskin tarvitsee käyttää ikinä. Kaikki säännöt sisältävät yhden terminaalisymbolin. Niiden tuottama merkkijono sijoitetaan synteettiseen attribuuttiin *w*. Koska operaattorisymbolin säännöt ovat suhteellisen yksikertaiset, ne on annettu sanallisen selittämisen sijaan taulukossa 9.

30	O → =	w(O) = =
31	O → !=	w(O) = <>
32	O → >	w(O) = >
33	O → <	w(O) = <
34	O → >=	w(O) = >=
35	O → <=	w(O) = <=

Taulukko 9. Operaattoriin sovellettavat säännöt

4.6.12 Arvo (V)

Symbolilla V kuvataan arvoa, johon attribuuttia verrataan XPath-ilmaisussa jollain vertailuoperaattorilla. Symboliin V voi soveltaa kahta eri sääntöä, jotka on kuvattu taulukossa 10. Säännöt vastaavat XPathin tukemia tietotyyppejä merkkijono, numeeriset arvot ja totuusarvot. Molemmissa säännöissä arvotetaan ainoastaan synteettinen attribuutti *q*. Säännössä 52 säännön oikealla puolella esiintyy kolme terminaalisymbolia: lainausmerkit, joiden sisällä on merkkijono. Säännön 52 yhteydessä XPath-ilmaisun sisältämä merkkijono muuttuu Cypher-kyselyyn sijoitettaessa vain lainausmerkkien osalta. Säännössä 53 taas terminaalisymboli kuvaa numeerista arvoa tai totuusarvoa. Säännössä 53 XPath-ilmaisun arvo siirtyy semanttisissa säännöissä attribuutin *q* arvoksi sellaisenaan, sillä Cypherissä numeeristen arvojen tai totuusarvojen kuvaamiseen ei tarvita ylimääräisiä merkkejä.

52	V → ”*string*”	q(V) = ‘*string’*
53	V → *numberOrBool*	q(V) = *numberOrBool*

Taulukko 10. Symbolin V kontekstivapaa kielioppi ja semanttiset säännöt

5 Esimerkit

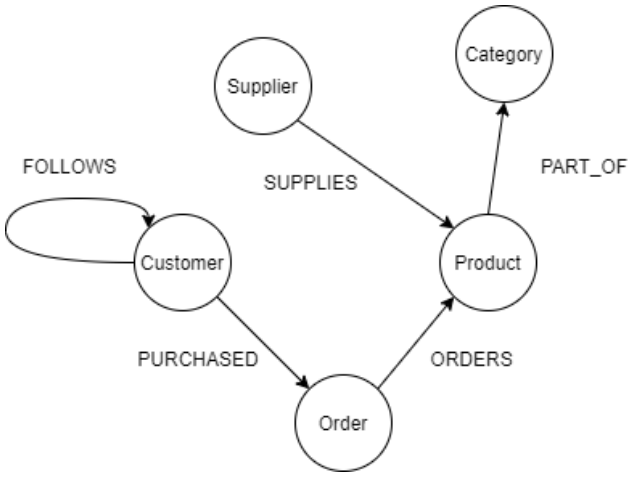
Tässä luvussa havainnollistetaan käytännön esimerkein XPath-ilmaisujen kääntäminen Cypher-ilmaisuiksi. Käännösesimerkeissä käydään yksityiskohtaisesti läpi aliluvussa 4.3 esitelty käännösprosessi käyttäen aliluvussa 4.6 esiteltyä attribuuttikielioppia. Attribuuttikielioppiin kuuluvaa kontekstivapaata kielioppia käytetään jäsentämään XPath-ilmaisu osiin, ja muodostamaan jäsennyspuu. Jäsennyspuussa kulkemisessa tuotetaan semanttiset säännöt XPath-ilmaisun osille käyttäen attribuuttikieliopin semanttisia sääntöjä.

Jäsennysesimerkki aliluvussa 5.1 on yksinkertainen ja kuvaa jäsentämisen ideaa. Aliluvussa 5.2 esitetään valikoituja XPath-kyselyitä ja niiden käännöksiä Cypherillä. Alilukujen 5.1 ja 5.2 yksinkertaisten esimerkkien täydentämiseksi laajempi jäsennysesimerkki esitetään liitteessä 3. Laajemman jäsennysesimerkin kysely on pidempi, ja siihen sisältyy monipuolisemmin XPathin tarjoamia ominaisuuksia graafiympäristössä.

Tämän luvun jäsennysesimerkissä ja esimerkkikyselyissä, sekä liitteen 3 laajemmassa jäsennysesimerkissä käytetään liitteen 1 mukaista taulukkomaista esitystapaa semanttisten sääntöjen muodostamiselle. Taulukossa vasemmanpuoleisemmassa sarakkeessa on säännön numero. Keskimmäisessä sarakkeessa on kontekstivapaan kieliopin sääntö. Attribuuttikieliopin semanttiset säännöt esitetään oikeinpuolimmaisimmassa sarakkeessa.

Jäsennysesimerkkien XPath-kyselyt perustuvat työssä käytettävään testitietokantaan. Kyselyissä käytetty testitietokanta Neo4J:ssä on Northwind-tietokanta. Tietokannan voi asentaa Neo4J-tietokannassa sivulla *www.neo4j.com/developer/example-data* olevien ohjeiden mukaan tai kirjoittamalla Neo4J Browserin tekstikenttään komennon `:play northwind-graph`. Northwind-tietokannan kaavio on esitetty kuvassa 5. Northwind-tietokannan kaaviossa kuvaillaan kuinka asiakas voi tehdä tilauksen. Tilaukseen voi kuulua useita tuotteita. Asiakas voi myös tehdä useita tilauksia. Tuotteella on valmistaja, ja sen on valmistanut valmistaja. Tietokanta koostuu entiteettityypeistä tuote (Product), kategoria (Category), tavarantoimittaja (Supplier), asiakas (Customer) ja tilaus (Order). Tietokannan suhteita ovat osa (Part_of), ”toimittaa” (Supplies), ”osti” (Purchased), ”tilaa”. Alkuperäistä Northwind-tietokantaa on laajennettu henkilöiden väliset seuraamissuhteet

(Follows), joka mahdollistaa transitiiviset suhteet asiakkaiden välillä. Luontilauseet näiden suhteiden luomiseen on liitteessä 2.



Kuva 5. Testitietokannan kaavio

5.1 Jäsennysesimerkki

Tämän aliluvun jäsennysesimerkin kysely hakee kaikki tietokannassa olevat tilaukset. Kysely 3 esittää, miten kuvaus ilmaistaan XPath-ilmaisuna.

```
*/PURCHASED
```

Kysely 3. XPath-kysely, joka hakee kaikki PURCHASED-nimiset suhteet

Tilaus on mallinnettu tietokannassa suhteena. Suhdetta graafissa kuvaavien kaarten lähtösolmu graafissa on aina tyyppiä asiakas, mutta sitä ei ole XPath-ilmaisussa erikseen määritelty. Tämän takia ilmaisussa käytetään alussa villiä korttia hakemaan kaikki solmut riippumatta entiteettityypistä.

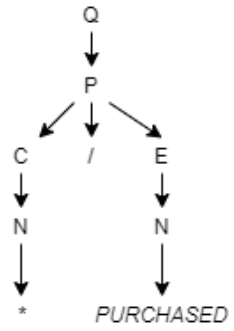
Kuten luvussa 4.3 käytiin läpi kääntämisen vaiheet, ensimmäisessä vaiheessa suoritetaan leksikaalinen analyysi. Analyysin tuloksena on alkionimien sekvenssi. Kyselyn 3 jakaminen tekstialkioihin on kuvattu taulukossa 11, missä on tunnistettu XPath-ilmaisun pienimmät tekstialkiot. Kullekin tekstialkiolle on annettu alkionimi, jotka perustuvat työn attribuuttikielioppiin.

Tekstialkio	**Alkionimi**
*	N
/	P
PURCHASED	N

Taulukko 11. Esimerkkikyselyn jaottelu alkionimiin

Käännöksen toisessa vaiheessa alkionimien sekvenssistä muodostetaan jäsennyspuu. Kyselyn 3 tuottama jäsennyspuu attribuuttikieliopilla AG_{Xypher} on esitetty kuvassa 6. Kuvan 6 jäsennyspuussa terminaalisymbolit ovat siis jäsennyspuun lehtisolmuja ja esi-

tetty puussa kursivoituna. Puun sisäsolmut ovat kontekstivapaan kieliopin $G_{PgXPath}$ nonterminaalisymbolien nimiä. Kuvassa ilmenee myös leksikaalisen tason yhteydessä esiteltyjen käsitteiden tekstialkioiden ja alkionimien yhteys toisiinsa. Tekstialkiot ovat aina jäsennyspuun lehtisolmuja ja alkionimi on lehtisolmujen vanhempi-solmu.



Kuva 6. Jäsennysesimerkin 1 jäsennyspuu

Kun jäsennyspuu on luotu, suoritetaan semanttinen analyysi. Semanttisessa analyysissa muodostetaan semantiikka kulkemalla jäsennyspuun solmut läpi. Koska työssä käytetään L-attribuoitua attribuuttikielioppia, jäsennyspuu käydään läpi syvyyshaulla valiten lapsisolmuista aina vasemmanpuoleisin solmu. Jäsennyspuun aloitussymboli on Q, joten semanttinen analyysi alkaa symbolista Q ja loppuu, kun puun prosessoinnissa on palattu symboliin Q. Käytettävä semanttinen sääntö määräytyy aina sen mukaan, mitä solmua prosessoidaan, ja mitkä ovat sen lapsisolmut jäsennyspuussa.

Ensimmäiseksi puussa siirrytään symbolista Q sen vasemmanpuoleiseen lapseen, joka on symboli P. Symboli P saa symbolilta Q periytyneiden attribuuttien *andInh*, *appliesTo*, *finishPath*, *insidePredicate* ja *pathInh* arvot. Nämä attribuutit arvotetaan joko tyhjäksi tai epätodeksi, koska muodostetussa Cypher-kyselyssä ei vielä tässä vaiheessa ole sisältöä. Alla on listattu symbolin P periytyneet attribuutit arvoineen.

1	$Q \rightarrow P$	**Periytyneet attribuutit** andInh(P) = NULL appliesTo(P) = NULL finishPath(P) = <u>false</u> insidePredicate = <u>false</u> pathInh(P) = NULL

Koska kyselyssä esiintyy vain kaksi askelta, joita erottaa terminaalisymboli /, sovelletaan attribuuttikieliopin sääntöä 7. Tällöin XPath-ilmaisun ensimmäinen askel tulkitaan solmuaskeleksi C ja toinen kaariaskeleksi E. Koska työssä sovelletaan L-attribuoitua

kielioppia, edetään ensin symboliin C, joka on jäsennyspuussa vasemmanpuoleisin lapsisolmu. Nonterminaalisymbolin E periytyneiden attribuuttien arvottaminen odottaa siis ensin symbolin C synteettisten attribuuttien arvottamista. Symbolille C periytyy attribuuttien *finishPath*, *andInh*, *insidePredicate* ja *pathInh* arvot. Attribuutti *finishPath* saa totuusarvoksi epätosi, koska sen arvo tulee tässä tapauksessa suoraan symbolilta Q. Periytyneen attribuutin *andInh* tyhjä arvo siirtyy myös suoraan symbolille C vastaavan attribuutin arvoksi. Alla on lueteltu kaikki symbolin C periytyneet attribuutit arvoineen.

7	P → C / E	**Periytyneet attribuutit** finishPath(C) = false andInh(C) = NULL pathInh(C) = NULL insidePredicate(C) = false

Koska solmuaskel C sisältää vain solmutestin, sovelletaan sääntöä 20. Säännön 20 mukaan säännön oikealla puolella nonterminaalisymboli N ei peri attribuutteja symbolilta C. Tällöin symbolin C perimien attribuuttien arvot eivät kulkeudu jäsennyspuussa alaspäin.

20	C → N	

Jäsennyspuussa symbolin N lapsisolmu on terminaalisymboli *, joten sovelletaan kieliopin sääntöä 28. Säännössä arvotetaan ainoastaan synteettinen attribuutti *q* ja *ret*. Nämä attribuutit arvotetaan uudella Cypher-muuttujalla `a0`. Muuttuja kuvaa Cypher-kyselyssä graafin rakennetta, joka tässä tapauksessa on solmu. Koska solmuaskelessa solmutestinä on villi kortti, mitään muuta tietoa ei tallenneta attribuutteihin.

28	N → *	**Synteettiset attribuutit** q(N) = ret(N) = a0

Koska säännössä 28 ei esiinny oikealla puolella nonterminaalisymboleita, käännetään kulkemissuunta jäsennyspuussa. Seuraavaksi palataan takaisin symbolin N vanhempisymboliin C ja sääntöön 20. Sen semanttisissa säännöissä symbolilta N kuvautunut uusi Cypher-muuttuja `a0` ympäröidään kaarisuluilla. Sulut kuvaavat Cypher-kyselyssä sen sisällön viittaavan solmuihin. Muuttujalla `a0` arvotetaan myös kyselyn paluuarvoa kuvaava

synteettinen attribuutti *ret*. Jos solmuaskel C olisi XPath-kyselyn viimeinen osa, niin solmuun liittyvään muuttujaan viitattaisiin `RETURN`-lausekkeessa paluuarvona.

20	C → N	**Synteettiset attribuutit** pq(C) = (a0) ret(C) = a0

Koska sääntö 20 ei sisällä vierailemattomia nonterminaalisymboleja säännön oikealla puolella, palataan symboliin P ja sääntöön 7. Tässä vaiheessa säännön 7 oikean puolen ensimmäisen symbolin C sisältämä puurakenne on käyty läpi. Tämä tarkoittaa, että symbolilta C kuvautuneilla synteettisten attribuuttien arvoilla voi nyt arvottaa seuraavien nonterminaalisymbolien periytyneet attribuutit tai symbolin P synteettiset attribuutit. Tässä tapauksessa niillä arvotetaan symbolin E periytyneet attribuutit. Yksi tällainen attribuutti on *appliesTo*, joka arvotetaan symbolilta C kuvautuneen synteettisen attribuutin *ret* arvolla `a0`. Lisäksi periytynyt attribuutti *andInh* arvotetaan symbolilta C kuvautuneen synteettisen attribuutin *andSyn* arvolla. Tämä arvo tosin on tyhjä, sillä symbolin C sisältämissä rakenteissa ei arvotettu Cypher-kyselyn `WHERE`-lausekkeeseen sijoitetavia kyselyn osia. Symboli E saa periytyneen attribuutin *pathInh* arvon suoraan symbolilta P, koska solmuaskelessa eli symbolissa C ei työn semanttisissa säännöissä voi luoda Cypher-kyselyyn kuuluvaa polun tunnusta. Lisäksi periytynyt attribuutti *insidePredicate* arvotetaan epätodeksi, koska XPath-kyselyn osa ei ole predikaatin sisällä. Alla on listattu symbolin E periytyneet attribuutit arvoineen.

7	P → C / E	**Periytyneet attribuutit** pathInh(E) = NULL appliesTo(E) = a0 andInh(E) = NULL insidePredicate(E) = <u>false</u>

Symbolille E sovelletaan kieliopin sääntöä 24, koska mukana on ainoastaan solmutestiä kuvaava nonterminaalisymboli N. Symboli N ei peri attribuutteja symbolilta E, joten symbolin E perimien attribuuttien arvot eivät kulkeudu jäsennyspuussa alaspäin.

20	E → N	

Symboliin N liittyy terminaalisymboli. Terminaalisymboli kuvaa tässä esimerkissä merkkijonoa PURCHASED, joka on XPathin askelen solmutestin nimi. Koska solmutestin nimi on merkkijono, sovelletaan symboliin N sääntöä 27. Merkkijonolla arvotetaan synteettinen attribuutti *name*. Synteettisen attribuutin *ret* arvoksi tulee uusi Cypher-muuttuja `a1`, joka siis on uniikki tunnus. Cypher-kyselyn `MATCH`-lausekkeeseen sijoitettava synteettinen attribuutti *q* arvotetaan juuri luodulla Cypher-muuttujalla `a1` ja solmutestin nimellä PURCHASED. Näitä erottaa toisistaan Cypherin syntaksissa kaksoispiste.

27	N → *name*	**Synteettiset attribuutit** q(N) = a1: PURCHASED ret(N) = a1 name(N) = PURCHASED

Säännössä 27 ei ole enää jäljellä vierailemattomia nonterminaalisymboleja, joten palataan takaisin symboliin E ja sääntöön 24. Säännön 24 semanttisissa säännöissä arvotetaan synteettiset attribuutit *pq* ja *ret*. Attribuutti *pq* arvotetaan symbolin N synteettisen *q* attribuutin arvolla. Koska sääntöä 24 käytetään kaariaskelen yhteydessä, synteettisen attribuutin *q* arvon ympärille sijoitetaan Cypherin kaarta kuvaavat merkit. Kaari osoittaa vasemmalta oikealle, koska säännössä 24 kuvaamalle kaariaskeleelle ei ole eksplisiittisesti määritetty akselia XPath-ilmaisussa. Tällöin akseli on XPathissa oletuksena *child*, ja käytetään taulukon 4 vastaavuutta muodostamaan kaari Cypher-kyselyyn. Synteettisen attribuutin *ret* arvo kuvautuu suoraan symbolin N vastaavalta attribuutilta.

24	E → N	**Synteettiset attribuutit** pq(E) = -[a1:PURCHASED]-> ret(E) = a1

Säännöllä 24 ei ole enää vierailemattomia symboleita, joten palataan symboliin P ja sääntöön 7. Nyt säännön 7 molempien nonterminaalisymbolien synteettiset attribuutit on arvotettu, joten voidaan arvottaa loputkin synteettiset attribuutit symbolille P. Synteettinen attribuutti *pq* arvotetaan symboleilta C ja E kuvautuneen vastaavan attribuutin arvolla. Arvottaminen tapahtuu asettelemalla attribuutin arvot peräkkäin tässä järjestyksessä. Synteettisen attribuutin *pq* arvon perään lisätään tyhjä sulkupari, koska lopulta attribuutin *pq* arvo sijoitetaan Cypher-kyselyssä `MATCH`-lausekkeeseen eikä `MATCH`-lau-

seke ei voi päättyä kaareen, vaikka XPath-ilmaisu päättyi kaariaskeleeseen. Koska kyselyssä ei ollut predikaatteja, jotka olisivat voineet sisältää sisäkkäisiä ilmaisuja tai suodattimia, synteettiset attribuutit *q* ja *w* arvotetaan säännössä tyhjällä arvolla. Koska ilmaisun kaariaskel E ei sisältänyt transitiivista akselia, synteettinen attribuutti *pathSyn* arvotetaan myös tyhjäksi. Koska symboli E on säännön lukusuunnassa viimeinen symboli, sen synteettisen attribuutin *ret* arvo `a1` on koko Cypher-ilmaisun paluuarvo.

7	P → C / E	**Synteettiset attribuutit** pq(P) = (a0)-[a1:PURCHASED]->() q(P) = NULL w(P) = NULL ret(P) = a1 pathSyn(P) = NULL

Kun symbolin P synteettiset attribuutit on arvotettu, palataan symboliin Q ja sääntöön 1. Säännössä 1 arvotetaan synteettinen attribuutti *ret*, joka kuvaa tässä säännössä koko Cypher-kyselyä. Symbolissa Q lisätään synteettisen attribuutin *pq* arvo Cypherin avainsanan `MATCH` perään. Koska symbolin P synteettinen attribuutti *q* oli tyhjä, ei avainsanan `MATCH` perään lisätä muuta. Koska symbolilta P kuvautuneiden synteettisten attribuuttien *w, with* ja *u* arvo on tyhjä, Cypher-kyselyyn ei lisätä lausekkeita `WHERE`, `WITH` tai `UNWIND`. `RETURN`-lausekkeeseen asetetaan avainsanan `RETURN` perään synteettisen attribuutin *ret* arvo `a1`.

1	Q → P	**Synteettiset attribuutit** ret(Q) = MATCH (a0)-[a1:PURCHASED]->() ret(Q) += RETURN a1

Kun on palattu lähtösymboliin Q, sen synteettisten attribuuttien arvottamisen jälkeen on muodostettu lopullinen Cypher-kysely. Lopullinen Cypher-kysely on siis juurisymbolin Q semanttisten sääntöjen synteettisen attribuutin *ret* arvo. Arvo on ilmaistu kyselyssä 4.

```
MATCH (a0)-[a1:PURCHASED]->()
RETURN a1
```

Kysely 4. Jäsennysesimerkin XPath-ilmaisua vastaava Cypher-ilmaisu

5.2 Esimerkkikyselyitä

Tässä luvussa esitellään testitietokannasta tehtyjä esimerkkikyselyitä XPathilla ja pohditaan XPathin ja Cypherin ominaisuuksia graafitietokannoissa. Esimerkkikyselyt esitetään liitteessä 4. Esimerkeissä esitetään kyselyn kuvaus, XPath-ilmaisu, ja tuotettu käännös Cypher-kielellä. Kyselyt havainnollistavat yksinkertaisten kyselyiden lisäksi Woodin [2012] graafitietokantojen yhteydessä tärkeiksi määrittelemiä kyselytyyppejä. Näitä ovat aggregaattikysely, konjunktiivinen kysely, RPQ-kysely ja kaksisuuntainen RPQ-kysely. Lisäksi verrataan Libkinin ja muiden [2016] mainitsemaa RPQ-kyselyä vertailuoperaatioiden kanssa, jotka yhdistävät graafissa navigoinnin vertailuoperaatioihin. Kyselyiden kieliasussa on noudatettu Cypherin dokumentaation suosituksia.

Esimerkeissä voi havaita sen, että XPathin etu Cypheriin nähden on lyhyissä kyselyissä. Esimerkiksi kyselyssä 7 XPathilla riittää ainoastaan entiteettityypin kirjoittaminen kaikkien Customer-solmujen palauttamiseksi. XPath-ilmaisu on täten erittäin intuitiivinen. Cypherillä täytyy määrittää erikseen lausekkeet sekä palautettava arvo, joka pidentää kyselyn pituutta.

Myös konjunktiiviset kyselyt voidaan ilmaista XPathilla Cypheriin nähden suoraviivaisemmin. Kyselyssä 9 XPathilla konjunktiivinen kysely kirjoitetaan yhden predikaatin sisälle and-operaattorilla eroteltuna. Tällöin kaikki and-operaattorien sitomat osat on sidottu predikaattia edeltäneeseen muuttujaan, joka tuntuu luontevalta. Lisäksi kyselyssä attribuuttien arvolla suodattaminen osana kyselyä tekee kyselystä suoraviivaisen. Cypherillä konjunktio toteutetaan useilla toisistaan erotetuilla poluilla. Jokaisesta polusta viitataan aiemmin esiteltyihin muuttujiin. Tämä vaatii tarkkuutta kyselyä kirjoittaessa. Yhtäsuuruusvertailut solmujen yhteydessä voidaan ilmaista Cypherissä intuitiivisesti. Cypherin ja XPathin tapa tehdä yhtäsuuruusvertailu myös kaaren yhteydessä on samankaltainen, kuten kyselyssä 12a esitetään. Kyselyssä 12b esitetään sama kysely kuin kyselyssä 12a, mutta sillä erotuksella, että yhtäsuuruuden lisäksi tarkastellaan erisuuruutta. Muiden vertailuoperaatioiden kuin yhtäsuuruusvertailujen kanssa Cypher ei ole enää yhä suoraviivainen kuin kyselyssä 12a. Cypher ei anna sijoittaa muita vertailuoperaatioita kaaren yhteyteen. Sen sijaan vertailuoperaatio suoritetaan kyselyn `WHERE`-osassa iteroimalla polkujen sisältämiä suhteita, ja suodattamalla siitä halutut tulokset. Tämän takia Cypher-kysely vaatii graafihahmon sitomista polkumuuttujaan. Tämä pidentää kyselyä huomattavasti ja on ehkä hieman epäintuitiivinen tapa XPathiin nähden, jossa jokainen vertailuoperaatio voidaan liittää kaariaskeleeseen.

Cypherin etuna on navigoinnin ilmaiseminen. Kyselyssä 10 XPath ilmaisee transitiiviset suhteet eli navigoimisen kaaria sen suuntaan *descendant*-akselilla. Cypher ilmaisee tämän suhteellisen suoraviivaisesti lisäämällä asteriskin kaaren nimen perään. Navigoinnissa Cypherin etu on myös kaksisuuntaisen navigoinnin ilmaiseminen. Kyselyssä 11 XPathilla ilmaistaan navigoiminen FOLLOWS-nimistä kaarta pitkin niin monta kertaa käänteiseen suuntaan kuin mahdollista ancestor-akselilla. Cypherin tapa ilmaista navigointi kaaren käänteiseen suuntaan on XPathiin nähden lyhyempi ja ymmärrettävämpi.

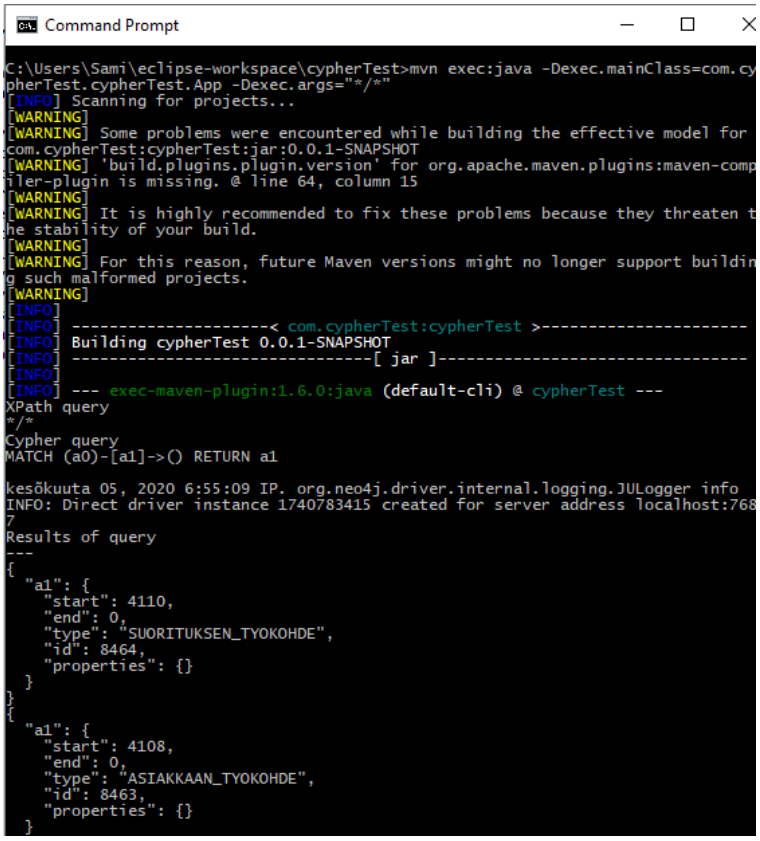
6 Kääntäjän toteutus

Työtä varten on toteutettu ohjelma Xypher [GitHubB, 2020], joka kääntää XPath-ilmaisut Cypher-kielen ilmaisuiksi. Kääntäjä on kirjoitettu Javalla. Kääntäjä käyttää ANTLR4-kirjastoa [ANTLR4] syötteenä saadun XPath-kyselyn prosessoimiseen leksikaalisella, syntaktisella ja semanttisella tasolla. ANTLR4 käyttää g4-tyyppistä kielioppitiedostoa luodakseen valmiin lähdekoodin kielen leksikaalista ja syntaktista analyysiä varten. Ohjelmoijan tehtäväksi jää ainoastaan semanttisten sääntöjen luonti kirjaston tuottamaan lähdekoodiin. Ohjelmassa käytetty kielioppi [GitHubA, 2020] sisältää XPath 1.0:n kieliopin. ANTLR4 muodostaa kieliopin mukaan lähdekoodin, jossa esiintyy metodit kullekin kieliopissa määritellylle symbolille. Lähdekoodin metodit saavat parametrina kontekstiobjektin. Tämä objekti sisältää tietoa jäsennyspuussa jo vierailluista solmuista, jolloin jäsennyspuussa etenemistä voidaan seurata.

Xypher on komentorivipohjainen ohjelma ilman graafisia käyttöliittymäelementtejä. Kuvassa 7 on kuvattu ohjelman suoritus komentoikkunassa. Ohjelman suorittamiseen ohjeet löytyvät GitHubista [GitHubB, 2020]. Toimiakseen ohjelma ottaa vastaan vähintään kolme parametria. Parametrit annetaan järjestyksessä XPath-kysely, Neo4J-tietokannan nimi, ja Neo4J-tietokannan salasana. Neljäntenä valinnaisena parametrina voi antaa Neo4J-tietokannan osoitteen. Parametrit kirjoitetaan lainausmerkkien sisälle välimerkein eroteltuna. Ohjelman suoritus onnistuu myös moderneissa ohjelmistoympäristöissä, kuten Eclipsessa.

Semanttisia sääntöjä käyttäen XPath käännetään Cypher-kyselykielen ilmaisuksi. Kun käännös on valmis, tulostetaan näytölle alkuperäinen XPath-ilmaisu ja tuotettu Cypher-kysely. Käännöksen jälkeen ohjelma avaa tietokantayhteyden ohjelman parametrien mukaiseen Neo4J-tietokantaan. Yhteys perustuu Bolt-protokollaan, joka on erityisesti Neo4J-tietokantaa varten toteutettu protokolla [Neo4J, 2020]. Kun yhteys on auki,

Neo4J vastaanottaa XPath-ilmaisusta käännetyn Cypher-kyselyn. Neo4J suorittaa itsenäisesti uudelleen luvussa 4.3 kuvatut käännöksen vaiheet, jotta kyselyllä voidaan hakea Cypher-kyselyn semantiikkaa vastaavat kartoitukset tietokannasta. Kyselyn tulokset palautetaan ohjelmaan, ja ne muotoillaan ohjelmassa JSON-objekteiksi ja tulostetaan. Palautettavat JSON-objektit näkyvät kuvassa 7.



Kuva 7. Xypher-ohjelman toiminta komentoikkunassa

7 Johtopäätökset

Tässä työssä on osoitettu, että XPath soveltuu graafikyselykieleksi luomalla semantiikka Cypher-kielen ilmaisuna. Myös kääntäjän toteuttaminen on teknisesti mahdollista. Sen lisäksi, että käännös onnistuu ja XPathille voidaan luoda semantiikka graafikyselykielenä, XPath tuo mukanaan selkeitä etuja Cypheriin nähden. Näistä kahdesta XPath on tunnetumpi kieli ja sen syntaksi muistuttaa monille käyttäjille tuttuja käytäntöjä muista entuudestaan tutuista ympäristöistä. Syntaksissa XPathin etu Cypheriin nähden on tiivis syntaksi. Tiiviimpi syntaksi johtuu osittain siitä, ettei XPathissa esiinny lausekkeita kyselyn eri toiminnoille, kuten Cypherin syntaksissa käytetyt lausekkeiden (`MATCH`, `WHERE` jne.) nimet, jotka sitovat itseensä kyselyn osat. Parhaimmillaan tämä ilmenee ly-

hyissä kyselyissä, kuten tietyn entiteettityypin entiteetit palauttavissa kyselyissä. Tällaisissa tapauksissa käyttäjän ei tarvitse kirjoittaa XPath-ilmaisuun mitään muuta kuin entiteettityypin nimi. XPathin syntaksissa monet ominaisuudet pystytään ilmaisemaan hyvinkin korkealla tasolla. XPathin implisiittistä luonnetta korostaa myös muuttujien puute kielestä. Muuttujiin viitataan XPathissa implisiittisesti, kun taas Cypherissä muuttujat kirjoitetaan kyselyyn eksplisiittisesti. XPathin predikaattien yhteydessä on intuitiivisesti helppo ymmärtää mihin predikaatti on sidottu. Cypherissä ehtojen sitominen vaatii muuttujiin viittaamista, joka vaatii tarkkuutta kyselyä kirjoitettaessa. XPathin implisiittisellä muuttujiin viittaamisella on myös selkeä haittansa, sillä muuttujiin ristiin viittaaminen kyselyn myöhemmissä vaiheissa on käytännössä mahdotonta XPathissa. Tietyissä kyselyissä tämä ominaisuus olisi hyödyksi. Tässä mielessä Cypher on monipuolisempi.

XPathin syntaksi mahdollistaa koko graafihahmon määrittelemisen kompaktilla tavalla kyselyyn. XPathissa graafihahmoissa voidaan erotella navigoinnilliset elementit näiden sisältämästä datasta. Tätä voidaan pitää tärkeänä erotteluna, joka selventää kyselyä. XPathin sisäkkäiset polkurakenteet mallintavat ilmaisuvoimaisesti hahmoon kuuluvien polkujen haarautumista, ja on logiikaltaan suoraviivaisempi Cypheriin nähden. Sisäkkäiset polkurakenteet ja ehdot ilmaistaan predikaateilla, jota ovat tasavertaisia siinä mielessä, että kaikki ehdot voidaan esittää XPathissa samalla tavalla. Cypherissä hahmon sisältämät polut ja vertailuoperaatiot tulee sijoittaa erikseen kyselyn eri lausekkeisiin. Tämän takia Cypherissä lisätyötä käyttäjälle aiheuttaa oikeisiin muuttujiin viittaaminen kyselyssä. Lisäksi predikaattien sisältämien vertailuoperaatioiden suhteen XPath on joustavampi ja suoraviivaisempi. XPathin predikaatin sisällä oleva vertailuoperaatio voi sisältää minkä tahansa operaattorin. XPathin syntaksissa ei ole eroa käyttääkö vertailuoperaatiota solmun vai kaaren yhteydessä, ja mikäli kaariaskelen akseli on transitiivinen tai ei. Cypherissä taas joillain kyselyillä, kuten havainnollistettuna aliluvussa 5.2, suodatus transitiivisen suhteen ehtojen suhteen voi johtaa pitkään ja monimutkaiseen kyselyyn. Tällaisessa tapauksessa Cypher-kyselyyn täytyy lisätä uusia lausekkeita, joiden merkitys saattaa olla epäselvä käyttäjille. Vaikka Cypherin dokumentaatio on ajantasainen ja laaja, eri lausekkeiden semantiikka voi vaatia käyttäjältä opettelemista. Tällaisilla kyselyillä Cypher menettää ilmaisuvoimaansa ja alkaa muistuttamaan matalamman tason kieliä, kun kysely sisältää iteraattoreita ja funktiota. Tietenkin näillä ominaisuuksilla voidaan Cypherissä saavuttaa hyvinkin monipuolisia kyselyitä graafitietokannasta, joihin XPath ei taivu. XPathissa samanlaiset ominaisuudet voidaan ilmaista yksinkertaisemmin ja kor-

keammalla tasolla. Toisin sanoen XPathissa samat Cypherin tarjoamat työkalut ominaisuuden ilmaisemiseen on ilmaisussa piilossa, joka luonnollisesti tekee kyselyistä helpompia. XPathin ilmaisuvoimaa voikin luonnehtia tietyissä kyselyissä paremmaksi graafitietokannassa kuin Cypherin. Tätä kuvastaa työn attribuuttikielioppi, jossa esimerkiksi toisiaan muistuttavien XPath-ilmaisujen semantiikka Cypher-kielellä eroaa huomattavasti toisistaan. Koska XPathin syntaksi muistuttaa predikaattilogiikkaa, tämä kuvastaa sitä, että tietyt ideat ovat helpommin ilmaistavissa XPathilla kuin Cypherillä. Muutenkin XPathin vahvimpana puolena näyttäytyy sen syntaksi, joka antaa myös graafiympäristössä määrittää polulle askeleita ja suodattimia sitä mukaa kun kyselyä luodaan.

Selkeitä Cypherin etuja ovat graafimaisuutta ilmentävä syntaksi, joka tekee kyselystä paremmin luettavan. XPathin syntaksi on tiivis ja intuitiivinen, mutta menettää kyselyn luettavuutta, kun semantiikassa solmut ja kaaret näyttävät kyselyn syntaksissa samanlaisilta. Tämä ei tarkoita, että Cypher-kyselyt olisivat aina siistityn näköisiä. Kuten aliluvun 5.2 esimerkeistä kävi ilmi, Cypher-kysely voi muuttua sekavaksi kokonaisuudeksi, kun kyselyssä esiintyy useita lausekkeita. Graafimaisuuden lisäksi Cypherillä voi myös asettaa usean muuttujan kyselyn paluuarvoksi, joka monipuolistaa kyselyn tuloksia. XPathissa voi asettaa vain yhden paluuarvon kyselylle. Toisaalta tämä tiivistää kyselyä, mutta usein kyselyissä hyödyllistä olisi palauttaa useampi kuin yksi paluuarvon määräämä tulosjoukko. Usean paluuarvon lisäksi Cypherillä voi navigoida useaan suuntaan. Cypher mahdollistaa navigoinnin graafissa määrittelemättä kaaren suuntaa, mikä on hyödyllinen ominaisuus, sillä kyselyjen kirjoittamien ei tällöin vaadi ennakkoon tietoa graafin rakenteesta.

Koska Cypher on erityisesti graafiympäristöön kehitetty kieli, voi todeta, että sillä on luonnollisestikin laajempi toiminnallisuus XPathiin nähden. Paikoin Cypherin ilmaisuvoima graafitietokannassa on parempi kuin XPathilla. Toisaalta Cypher ei rajoita XPathin toiminnallisuutta hahmontäsmäyksen, graafissa navigoinnin, tulosten suodattamisen tai kyselyn projektion kannalta.

Jatkokehitys on tässä työssä esitellyn attribuuttikieliopin pohjalta mahdollista. Tässä työssä esitelty kääntäjä on melko alkeellinen vuorovaikutuksensa osalta. Kääntäjää olisi mahdollista kehittää graafisella käyttöliittymällä, jolla voisi pyrkiä selkiyttämään visuaalisin keinoin esimerkiksi tässä työssä epäselväksi havaittujen solmuaskelten ja kaariaskelten eroa. XPathin käytön tulisi olla tehokasta myös graafitietokannoissa, joten aikavaatimuksen tarkastelu tulisi olla oleellisimpia jatkokehityksen kohteita. Kun XPathia

käyttää Neo4J-tietokantaan käännettynä Cypher-kielen ilmaisuksi, koko käännösprosessin XPath-ilmaisun syöttämisestä alkaen tulisi olla mahdollisimman tehokas. Tässä työssä XPath on käännetty lisäksi melko naiivilla tavalla sitä semanttisesti vastaavaksi Cypher-ilmaisuksi. Siksi yksi tehokkuuteen liittyvä jatkokehityksen näkökulma olisi varmistaa, että käännös tuottaa sellaisen Cypher-ilmaisun, joka vastaa semantiikaltaan XPath-ilmaisua, mutta joka Neo4J-tietokannassa evaluoidaan mahdollisimman tehokkaasti. Esimerkiksi Holzschuher ja Peinl [2013] huomasivat, että Cypherin suoritusaika riippuu kyselyn muotoilusta.

Kuten luvussa 4 mainittiin, aliluvussa 4.6 määritetty attribuuttikielioppi ei kata koko XPath 1.0-versiota. Käännöksessä käytetyn XPathiin perustuvan kieliopin laajentaminen olisi luonnollinen jatkokehityksen aihe XPathin ilmaisuvoiman lisäämiseksi graafiympäristössä. Isoimpina puutteina tässä työn rajauksessa lienee XPathissa negaation ilmaisevan not-funktion ja loogisen disjunktion toteuttamatta jättäminen. Ne olisivat ilmaisuvoimaisia operaatioita, mutta toisaalta niiden semantiikka Cypherissä vaatii lisää tutkimista.

Tässä työssä esitellyt ideat XPathin semantiikasta Cypher-kieleksi käännettynä ja edelleen graafitietokantojen kyselykielenä eivät ole ainoita oikeita ratkaisuja. Työssä on pääasiassa käytetty XPathin version 1.0:n syntaksia laajentamatta sitä muutamaa aggregointifunktion nimeä lukuun ottamatta. Isoin keskustelun aihe jatkossa on miten graafitietokannan solmut ja kaaret olisi luontevinta esittää XPathissa. Tässä työssä ne on mallinnettu syntaksissa tasavertaisina XPathin elementtisolmuina. Tämä tuo ilmaisuvoimaa, koska kaaret ja solmut voidaan nimetä ja niihin sitoa predikaatteja. Samalla kuitenkin tällä tavalla menetetään ilmaisun luettavuutta, koska solmuilla ja kaarilla ei ole XPath-ilmaisussa syntaktista eroja. Myös XPath-ilmaisun evaluoinnin tuottamat tietotyypit on pyritty pitämään samanlaisena kääntäessä kielen ilmaisuja Cypherille. Työssä esimerkiksi solmujoukon palauttavat XPath-ilmaisut käännetään Cypheriksi siten, että Cypher-kysely palauttaa joukon objekteja, totuusarvon palauttavat XPath-ilmaisut käännetään Cypher-kyselyksi, joka palauttaa totuusarvon ja niin edelleen. Lopullisen Cypher-kyselyn evaluointia voisi joissain säännöissä pohtia. Esimerkiksi liitteessä 1 esitetyt attribuuttikieliopin säännöt, joissa säännöissä 11 ja 12 polun perässä olevaa attribuuttia verrataan johonkin arvoon. Tällöin XPath-kyselyssä semantiikka evaluoi aina ilmaisusta totuusarvon. Työn lähestymistavassa kysely tuottaa totuusarvon myös Cypher-kyselyssä. Totuusarvon sijaan ilmaisu voisi palauttaa solmujoukon niistä tuloksista, jotka täsmäävät ilmaisun hahmoon ja ilmaisun perässä olevaan vertailuun.

Tässä työssä käytetään XPathin akselien alkuperäisiä nimiä. Niiden käytöllä on helpompi tuottaa vastaavuus dokumentoidun XPathin version 1.0 osista Cypher-kielen osiin. Koska työssä sovelletaan XPathin virallisen dokumentaation mukaisia akselien nimiä, johtaa tämä joissain paikoissa pitkiin ilmaisuihin. Esimerkiksi *parent*-akselin kanssa ilmaisuvoima kärsii hieman, sillä navigoidessa akselilla kaarta sen käänteiseen suuntaan on solmutestin ja predikaatin ilmaiseminen hieman kömpelöä verrattuna *child*-akseliin, jolle akselia ei erikseen tarvitse kirjoittaa. Hyvänä vertailukohtana tähän on Cypherin syntaksi, jossa pienellä muutoksella kaaren suunnan saa kyselyssä vaihdettua. Akseleille voisi olla mahdollista kehittää lyhyemmät ja paremmin kuvaavat nimet jatkokehityksen puitteissa. Työssä esitelty navigoimisen idea perustuu oleellisimpiin graafissa tehtäviin operaatioihin. Esimerkiksi transitiivisuus esitettynä `//`-notaatiolla olettaa tässä työssä molempien puolten olevan graafitietokannan solmuja, koska solmujen välisen polun löytäminen on yksi graafien oleellisimpia ominaisuuksia. Toisaalta `//`-notaatiota voisi jatkokehityksessä harkita käytettävän lyhenteenä navigoinnille kaarten yhteydessä, ja antaa määrittää haettavan transitiivisen suhteen nimi notaation perään.

Lisäksi yhden mielenkiintoisen vertailukohdan XPathin käyttöön graafitietokannoissa toisi XPathin vertailu opittavuutensa osalta moderneihin graafikyselykieliin. Graafikyselykieliä on verrattu aiemmassa tutkimuksessa toisiinsa sekä perinteiseen relaatiotietokantojen SQL-kyselykieleen.

Viiteluettelo

Almabdy, Soad. 2018. Comparative analysis of relational and graph databases for social networks. In: *Proceedings of the 1st International Conference on Computer Applications & Information Security,* 1-4.

Angles, Renzo and Claudio Gutierrez. 2008. Survey of graph database models. *ACM Computing Surveys,* Vol 40, No 1, 1-39.

Angles, Renzo, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2017. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, Vol 50, No 5, 1-40.

Angles, Renzo. 2018. *The property graph database model*. Universidad de Talca. Department of Computer Science.

Angles, Renzo, Marcelo Arenas, Pablo Barceló, Peter Boncz, George Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan Sequenda, Oskar van Rest, and Hannes Voigt. 2018. G-CORE: a core for future graph query

language. In: *Proceedings of the 2018 International Conference on Management of Data*, 1421-1432.

Angles, Renzo, Harsh Thakkar, and Dominik Tomaszuk, 2020. Mapping RDF Databases to Property Graph Databases. *IEEE Access*, Vol 80, 86091-86110.

ANTLR4. https://www.antlr.org/. Luettu 28.4.2020.

Bagan, Guillaume, Angela Bonifati, and Benoit Groz. 2012. A trichotomy for regular simple path queries on graphs. *Journal of Computer and System Sciences*, Vol 108, 29-48.

Barceló, Pablo. 2013. Querying graph databases. In: *Proceedings of the 32nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 175-187.

Barceló, Pablo, Leonid Libkin, and Juan L. Reutter. 2014. Querying regular graph patterns. *Journal of the ACM*, Vol 61, No 1, 1-54.

Bárcenas, Everardo, Edgard Benítez-Guerrero, and Jesús Lavalle. 2015. On regular paths with counting and data tests. *Electronic Notes in Theoretical Computer* Science, Vol 328, 3-16.

Beaudou, Laurent, Florent Foucaud, Florent Madelaine, Lhouari Nourine, and Gaétan Richard. 2019. Complexity of Conjunctive Regular Path Query Homomorphisms In: *15th Conference on Computability in Europe,* 108-119*, Lecture Notes in Computer Science 11568*.

Benedikt, Michael and Christoph Koch. 2009. XPath leashed. *ACM Computing Surveys* Vol 41, No 1, 1-54.

Bergmann, Gábor, István Ráth, Tamás Szabó, Paolo Torrini, and Dániel Varró. 2012. Incremental pattern matching for the efficient computation of transitive closure. In: *Proceedings of the 6th International Conference on Graph Transformations*, *Lecture Notes in Computer Science 7562*, 386-400.

Bordoloi, Subhrajyoti, Deep Das, Sabiha Barlaskar, and Bichitri Kalita. 2014. Creating graph databases from relational databases: an implementation. *International Journal of Advanced Research in Computer Science*, Vol 5, No 7, 179-182.

Buneman, Peter. 1997. Semistructured data. In: *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems*, 117-121.

van Bruggen, Rik. *Learning Neo4j: Run Blazingly Fast Queries on Complex Graph Datasets with the Power of the Neo4j Graph Database*. Packt Publishing, 2014.

Calvanese, Diego, Moshe Vardi, Giuseppe de Giacomo, and Maurizio Lenzerini. 2000. View-based query processing for regular path queries with inverse. In: *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 58-66.

Calvanese, Diego, Moshe Vardi, Giuseppe de Giacomo, and Maurizio Lenzerini. 2003. Reasoning on Regular Path Queries. *ACM SIGMOD Record*, Vol 32, No 4, 83-92.

Cassidy, Steve. 2003. Generalizing XPath for directed graphs. In: *Proceedings of the Extreme Markup Languages, 2003.*

Codd, Edgar. 1970. A relational model for large shared data banks. *Communications of the ACM*, Vol 13, No 6, 377-387.

Colazzo, Dario and Carlo Sartiani. 2015. *Typing regular path query languages for data graphs*. Universite Paris-Dauphnine, Universita della Basilicata.

Cormen, Thomas. *Introduction to Algorithms*. MIT Press, 2001.

Cure, Olivier and Guillaume Blin. *RDF Database Systems; Triples Storage and SPARQL Query Processing.* Morgan Kaufmann, 2015.

Cypher. *https://neo4j.com/docs/cypher-manual/*. Luettu 28.4.2020.

DBEngines.com. *https://db-engines.com/en/ranking*. Luettu 18.2.2021.

Elmasri, Ramez and Shamkant Navathe. *Fundamentals of Database Systems*. Pearson Addison-Wesley, 2007.

Francis, Nadime, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In: *Proceedings of the 2018 International Conference on Management of Data*, 1433-1445.

Frisendal, Thomas. *Graph Data Modeling for NoSQL and SQL*. Technics Publications, 2016.

Gallagher, Brian. 2006. *Matching structure and semantics: A survey on graph-based pattern matching*. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.

GitHubA. *https://github.com/antlr/grammars-v4/blob/master/xpath1/xpath.g4*. Luettu 4.6.2020.

GitHubB. *https://github.com/SSSvensk/gradu*. Luettu 20.8.2020.

Gottlob, Georg, Christoph Koch, and Reinhard Pichler. 2003. XPath processing in a nutshell. *ACM SIGMOD Record*, Vol 32, No 1, 12-19.

Grust, Torsten, Maurice Keulen, and Jens Teubner. 2004. Accelerating XPath evaluation in any RDBMS. *ACM Transactions on Database Systems*, Vol 29, No 1, 91-131.

Gyssens, Marc, Jan Paredeans, Dirk Van Gucht, and George Fletcher. 2006. Structural characterizations of the semantics of XPath as navigation tool on a document. In: *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium of Database Systems*, 318-327.

Harrison, Guy. *Next Generation Databases NoSQL and Big Data*. Apress, 2015.

Harsu, Maarit. *Ohjelmointikielet*. Talentum, 2005.

Henglein, Fritz and Lasse Nielsen. 2011. Regular expression containment: coinductive axiomatization and computational interpretation. *ACM SIGPLAN Notices*, Vol 46, No 1, 385-398.

Holzner, Steven. *XPath: Navigating XML with XPath 1.0 and 2.0: Kick Start*. Sams, 2004.

Holzschuher, Florian and René Peinl. 2013. Performance of graph query languages: comparison of Cypher, Gremlin and native access in Neo4j. In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 195-204.

Holzschuher, Florian and René Peinl. 2016. Querying a graph database – language selection and performance considerations. *Journal of Computer and System Sciences*, Vol 82, No 1, 45-68.

Hopcroft, John and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

Karol, Sven. 2006. *An introduction to attribute grammars*. Technische Universität Dresden. Department of Computer Science.

Knuth, Donald. 1968. Semantics of context-free languages. *Theory of Computing System*, Vol 2, No 2, 127-145.

Koivisto, Pertti ja Riitta Niemistö. 2018. *Graafiteoriaa*. Tampereen Yliopisto. Informaatiotieteiden yksikkö.

Kostylev, Egor, Juan Reutter, and Domagoj Vrgoč. 2016. Static analysis of navigational XPath over graph databases. *Information Processing Letters*, Vol 116, No 7, 467-474.

Kostylev, Egor, Juan Reutter, and Domagoj Vrgoč. 2018. Containment of queries for graphs with data. *Journal of Computer and System Sciences*, Vol 92, 65-91.

Kumar, Rohit. 2015. Graph databases: a survey. In: *International Conference on Computing, Communication & Automation*, 785-790.

Libkin, Leonid and Domagoj Vrgoč. 2012. Regular path queries on graphs with data. In: *Proceedings of the 15th International Conference on Database Theory*, 74-85.

Libkin, Leonid, Wim Martens and Domagoj Vrgoč. 2013. Querying graph databases with XPath. In: *Proceedings of the 16th International Conference on Database Theory,* 129-140.

Libkin, Leonid, Wim Martens and Domagoj Vrgoč. 2016. Querying graphs with data. *Journal of the ACM*, Vol 63, No 2, 1-53.

Mami, Mohamed, Damien Graux, Harsh Thakkar, Simon Scerri, Sren Auer, and Jens Lehmann. 2019. *The Query translation landscape: a survey*. Enterprise Information Systems, Fraunhofer IAIS, St. Augustin & Dresden. ADAPT Centre, Trinity College of Dublin. Smart Data Analytics group, University of Bonn. TIB Leibniz Information Centre for Science and Technology. L3S Research Center, Leibniz University of Hannover.

Martens, Wim and Tina Trautner. 2017. *Enumeration problems for regular path queries*. University of Bayreuth.

Melton, Jim and Stephen Buxton. *Querying XML: XQuery, XPath, and SQL/XML in Context.* Morgan Kaufmann Publishers, 2006.

Needham, Mark. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, 2019.

Neo4j. *https://neo4j.com/*. Luettu 28.4.2020.

Olsson, Alexander and Therese Magnusson. 2018. *Implementing and evaluating a breadth-first search in Cypher*. Pro gradu -tutkielma. Lund University. Department of Computer Science.

Pettovello, Mark and Farshad Fotouhi. 2006. MTree: An XML Graph Index. In*: Proceedings of the ACM Symposium on Applied Computing*, 474-481.

RDF. 2014. *https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/Overview.html*. Luettu 28.4.2020.

Saake, Gunter, Kai-Uwe Sattler, und Andreas Heuer. *Datenbanken -- Konzepte und Sprachen*. Mitp Verlag, 2018.

SparQL. *https://www.w3.org/TR/sparql11-query/*. Luettu 16.3.2020.

Tiwari, Shashank. *Professional NoSQL*. Wrox, 2011.

Vainio, Johanna and Marko Junkkari. 2014. SQL-based semantics for path expressions over hierarchical data in relational databases. *Journal of Information Science*, Vol 40, No 3, 293-312.

Vardi, Moshe. 2016. A theory of regular queries. In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 1-9.

de Virgilio, Roberto, Antonio Maccioni, and Riccardo Torlone. 2013. Converting relational to graph databases. In: *First International Workshop on Graph Data Management Experiences and Systems*, 1-6.

Wood, Peter. 2012. Query languages for graph databases. *ACM SIGMOD Record*, Vol 41, No 1, 50-60.

XML. *https://www.w3.org/TR/REC-xml/*. Luettu 16.3.2020.

XPath. *https://www.w3.org/TR/1999/REC-xpath-19991116/*. Luettu 17.3.2020.

Liite 1: L-attribuoitu attribuuttikielioppi AG_{Xypher}

Id	Sääntö	Semanttiset säännöt
1	Q → P	**Periytyneet attribuutit** andInh(P) = NULL appliesTo(P) = NULL finishPath(P) = false insidePredicate(P) = false pathInh(P) = NULL **Synteettiset attribuutit** ret(Q) = **MATCH** pq(P) q(P) ret(Q) += **UNWIND** u(P) , if u(P) ≠ NULL ret(Q) += **WITH** with(P) , if withP) ≠ NULL ret(Q) += **WHERE** w(P), if w(P) ≠ NULL ret(Q) += **RETURN** ret(P)
2	Q → F	**Periytyneet attribuutit** andInh(F) = NULL appliesTo(F) = NULL finishPath(F) = false insidePredicate(F) = false pathInh(F) = NULL **Synteettiset attribuutit** ret(Q) = **MATCH** q(F) ret(Q) += **UNWIND** u(F) , if u(F) ≠ NULL ret(Q) += **WITH** with(P) , if withP) ≠ NULL ret(Q) += **WHERE** w(F) , if w(F) ≠ NULL ret(Q) += **RETURN** ret(F)
3	Q1 → P | Q2	**Periytyneet attribuutit** andInh(P) = NULL appliesTo(P) = NULL finishPath(P) = false insidePredicate(P) = false pathInh(P) = NULL **Synteettiset attribuutit** ret(Q1) = **MATCH** pq(P) q(P) ret(Q1) += **UNWIND** u(P) , if u(P) ≠ NULL ret(Q1) += **WITH** with(P) , if withP) ≠ NULL ret(Q1) += **WHERE** w(F) , if w(P) ≠ NULL ret(Q1) += **RETURN** ret(P) **UNION** ret(Q2)

4	Q1 → F | Q2	**Periytyneet attribuutit** andInh(F) = NULL appliesTo(F) = NULL finishPath(F) = false insidePredicate(F) = false pathInh(F) = NULL **Synteettiset attribuutit** ret(Q1) = **MATCH** q(F) ret(Q1) += **UNWIND** u(F) , if u(F) ≠ NULL ret(Q1) += **WITH** with(P) , if withP) ≠ NULL ret(Q1) += **WHERE** w(F) , if w(F) ≠ NULL ret(Q1) += **RETURN** ret(F) **UNION** ret(Q2)
5	F → *f_name*(P)	**Periytyneet attribuutit** andInh(P) = andInh(F) appliesTo(P) = appliesTo(F) finishPath(P) = finishPath(F) insidePredicate(P) = insidePredicate(F) pathInh(P) = pathInh(F) **Synteettiset attribuutit** q(F) = pq(P) q(P) u(F) = u(P) with(F) = with(P) w(F) = w(P) ret(F) = *f_name***(**ret(P)**)**
6	P1 → C / E / P2	**Periytyneet attribuutit** andInh(C) = andInh(P1) finishPath(C) = finishPath(P1) andInh(E) = andSyn(C) appliesTo(E) = ret(C) appliesTo(P2) = ret(E) andInh(P2) = andSyn(C) finishPath(P2) = finishPath(E) pathInh(P2) = pathSyn(E) insidePredicate(C) = insidePredicate(P1) insidePredicate(E) = insidePredicate(P1) insidePredicate(P2) = insidePredicate(P1) transitiveInh(P2) = transitiveSyn(E) appliesTo(P2) = ret(E) **Synteettiset attribuutit** pq(P1) = pq(C) pathSeq(E) pq(E) pq(P2)

		q(P1) = q(C) q(E) q(P2) u(P1) = u(P2) with(P1) = with(P2) w(P1) = w(C) w(E) w(P2) ret(P1) = ret(P2)
7	P → C / E	**Periytyneet attribuutit** finishPath(C) = finishPath(P1) andInh(C) = andInh(P) pathInh(C) = pathInh(P) pathInh(E) = pathInh(P) appliesTo(E) = ret(C) andInh(E) = andSyn(C) insidePredicate(C) = insidePredicate(P) insidePredicate(E) = insidePredicate(P) **Synteettiset attribuutit** pq(P) = pq(C) pathSeq(E) pq(E) **0** q(P) = q(C) q(E) w(P) = w(C) w(E) ret(P) = **relationships(**pathInh(E)**)** , if transitive(E) = true ret(P) = ret(E) else pathSyn(P) = pathSyn(E)
8	P → C	**Periytyneet attribuutit** andInh(C) = andInh(P) pathInh(C) = pathInh(P) insidePredicate(C) = insidePredicate(P) **Synteettiset attribuutit** pq(P) = pq(C) q(P) = q(C) w(P) = w(C) u(P) = u(C) with(P) = with(C) ret(P) = ret(C) andSyn(P) = andSyn(C)
9	P1 → C//P2	**Periytyneet attribuutit** finishPath(C) = finishPath(P1) pathInh(C) = pathInh(P1) pathInh(P2) = pathInh(P1) andInh(P2) = andSyn(C) insidePredicate(C) = insidePredicate(P1) insidePredicate(P2) = insidePredicate(P1)

		Synteettiset attribuutit pq(P1) = pq(C)**-[*]->**pq(P2) q(P1) = q(P) q(P2) u(P1) = u(P2) with(P1) = with(P2) w(P1) = w(C) w(P2) ret(P1) = ret(P2)
10	P1 → P2/A	**Periytyneet attribuutit** pathInh(P2) = pathInh(P1) andInh(P2) = andInh(P1) transitiveInh(P2) = transitiveSyn(P2) insidePredicate(P2) = insidePredicate(P1) **Synteettiset attribuutit** q(P1) = q(P2) pq(P1) = pq(P2) var = new_v() rel = new_v() u(P1) = **relationships(** pathSyn(P2) **) AS** var , if pathSyn(P2) ≠ NULL with(P1) = var **as** rel, if pathSyn(P2) ≠ NULL w(P1) = w(P2) , if insidePredicate w(P1) = w(P2) andSyn(P2) **EXISTS**(rel.ret(A)) if pathSyn(P2) ≠ NULL && insidePredicate w(P1) = w(P2) andSyn(P2) **EXISTS**(ret(P2).ret(A)) , if pathSyn(P2) == NULL && insideP-redicate ret(P1) = rel.ret(A) if pathSyn(P2) ≠ NULL ret(P1) = ret(P2) **.** ret(a) else
11	P1 → P2/A=V	**Periytyneet attribuutit** pathInh(P2) = pathInh(P1) andInh(P2) = andInh(P1) transitiveInh(P2) = transitiveSyn(P1) insidePredicate(P2) = insidePredicate(P1) **Synteettiset attribuutit** q(P1) = q(P2) pq(P1) = pq(P2) w(P1) = w(P2) andSyn(P) ALL(relationships(pathSyn(P2)) , if pathSyn(P2) ≠ NULL w(P1) = w(P2) andSyn(P) ret(P2).ret(A) = ret(V), if pathSyn(P2) == NULL var = new_v() rel = new_v() u(P1) = **relationships(** pathSyn(P2) **) AS** rel , if pathSyn(P2) ≠ NULL with(P1) = var **as** rel, if pathSyn(P2) ≠ NULL w(P1) = **ALL(rel in var** , if pathSyn(P2) ≠ NULL

		w(P1) += rel.ret(A) = ret(V)) ,if pathSyn(P2) ≠ NULL w(P1) = andSyn(P2) ret(P2).ret(A) = ret(V) , if pathSyn(P2) == NULL ret(P1) = count(var) > 0, if pathSyn(P2) ≠ NULL ret(P1) = count(ret(P2)) > 0, if pathSyn(P2) == NULL
12	P1 → P2/A O V	**Periytyneet attribuutit** pathInh(P2) = pathInh(P1) andInh(P2) = andInh(P1) transitiveInh(P2) = transitiveSyn(P1) insidePredicate(P2) = insidePredicate(P1) **Synteettiset attribuutit** q(P1) = q(P2) pq(P1) = pq(P2) w(P1) = w(P2) andSyn(P) ALL(relationships(pathSyn(P2)) , if pathSyn(P2) ≠ NULL w(P1) = w(P2) andSyn(P) ret(P2).ret(A) = ret(V), if pathSyn(P2) == NULL var = new_v() rel = new_v() u(P1) = **relationships(** pathSyn(P2) **) AS** var , if pathSyn(P2) ≠ NULL with(P1) = var **as** rel, if pathSyn(P2) ≠ NULL w(P1) = **ALL(rel in var** , if pathSyn(P2) ≠ NULL w(P1) += rel.ret(A) w(O) ret(V)) ,if pathSyn(P2) ≠ NULL w(P1) = andSyn(P2) ret(P2).ret(A) w(O) ret(V) , if pathSyn(P2) == NULL ret(P1) = count(var) > 0, if pathSyn(P2) ≠ NULL ret(P1) = count(ret(P2)) > 0, if pathSyn(P2) == NULL
13	P → A	**Synteettiset attribuutit** var = new_v() q(P) = **(** var **)** if insidePredicate(P) == FALSE ret(P) = var**.**ret(A) w(P) = **EXISTS**(appliesTo(P).ret(A)) , if insidePredicate(P)
14	P → A=V	**Synteettiset attribuutit** var = new_v() q(P) = **(** var {ret(A): ret(V)}**)** if appliesTo(P) == NULL w(P) = andInh(P) appliesTo(P) = ret(V) , if appliesTo(P) ≠ NULL and transitiveInh(P) == false w(PR) += **ALL(rel in relationships(** pathInh(P) **) WHERE** , if transitiveInh(P) w(PR) += **rel** , if transitiveInh(P) w(PR) += **.** ret(A) = q(V) if transitiveInh(P) == false w(PR) += **)** , if transitiveInh(P) andSyn(P) = **AND** ret(P) = count(var) > 0 , if appliesTo(P) == false ret(P) = count(appliesTo(P)) > 0, if appliesTo(P)

15	P → A O V	**Synteettiset attribuutit** var = new_v() q(P) = **(** new_v() **)** if appliesTo(P) == NULL w(P) = andInh(P) var.ret(A) w(O) ret(V) if appliesTo(P) == NULL w(P) = andInh(P) appliesTo(P).ret(A) w(O) ret(V) if appliesTo(P) ≠ NULL ret(P) = count(var) > 0 andSyn(P) = AND
16	PRP → E / P	**Periytyneet attribuutit** pathInh(E) = pathInh(PRP) andInh(E) = andInh(PRP) andInh(P) = andSyn(E) appliesTo(P) = ret(E) transitiveInh(P) = transitiveSyn(E) insidePredicate(E) = true insidePredicate(P) = true **Synteettiset attribuutit** q(PRP) = pq(E) pq(P) q(P) w(PRP) = w(E) w(P) andSyn(PRP) = andSyn(P)
17	C → AX::N[PR]	**Periytyneet attribuutit** appliesTo(PR) = ret(N) dotInh(PR) = NULL andInh(PR) = andInh(C) **Synteettiset attribuutit** pq(C) = **(** q(N) **{**equal(PR)**})** , if equal(PR) ≠ NULL pq(C) = **(** q(N) **)** else pq(C) += **,(**ret(N)**)** , if finishPath(C) q(C) = q(PR) w(C) = w(PR) ret(C) = ret(N) andSyn(C) = andSyn(PR)
18	C → AX::N	**Synteettiset attribuutit** pq(C) = **(** q(N) **)** pq(C) += **,(**ret(N)**)** , if finishPath(C) ret(C) = ret(N)
19	C → N[PR]	**Periytyneet attribuutit** appliesTo(PR) = ret(N) dotInh(PR) = NULL andInh(PR) = andInh(C) **Synteettiset attribuutit**

		pq(C) = **(** q(N) **{**equal(PR)**})** , if equal(PR) ≠ NULL pq(C) = **(** q(N) **)** else pq(C) += **,(**ret(N)**)** , if finishPath(C) q(C) = q(PR) w(C) = w(PR) ret(C) = ret(N) andSyn(C) = andSyn(PR)
20	C → N	**Synteettiset attribuutit** pq(C) = **(** q(N) **)** pq(C) += **,(**ret(N)**)** , if finishPath(C) ret(C) = ret(N)
21	E → AX::N[PR]	**Periytyneet attribuutit** appliesTo(PR) = ret(N) dotInh(PR) = NULL transitiveInh(PR) = transitiveSyn(AX) pathInh(PR) = pathSyn(E) andInh(PR) = andInh(E) insidePredicate(PR) = true **Synteettiset attribuutit** pq(E) = start(AX) **:** name(N) end(AX) , if transitiveSyn(AX) == true and equal(PR) == NULL pq(E) = start(AX) **:** name(N) **{**equal(PR) **}** end(AX) , if transitiveSyn(AX) == true and equal(PR) ≠ NULL pq(E) = start(AX) q(N) **{**equal(PR) **}** end(AX) , if transitiveSyn(AX) == false and equal(PR) ≠ NULL pq(E) = start(AX) q(N) end(AX) else q(E) = **(** q(PR) **)** w(E) = w(PR) ret(E) = ret(N) pathSyn(E) = new_v() pathSeq(E) =**,** pathSyn(E) **=(**appliesTo(E) **)** , if transitiveSyn(AX) == true andSyn(E) = andSyn(PR)
22	E → AX::N	**Synteettiset attribuutit** pq(E) = start(AX) **:** name(N) end(AX) , if transitiveSyn(AX) = true pq(E) = start(AX) q(N) end(AX) else pathSyn(E) = new_v() pathSeq(E) =**,** pathSyn(E) **=(**appliesTo(E) **)** , if transitiveSyn(AX) == true ret(E) = ret(N)
23	E → N[PR]	**Periytyneet attribuutit** appliesTo(PR) = ret(N)

		dotInh(PR) = NULL andInh(PR) = andInh(E) insidePredicate(PR) = true **Synteettiset attribuutit** pq(E) = **-[** q(N) pq(E) += **{** equal(PR) **}** , if (equal(PR) ≠ NULL) pq(E) += **]->** q(E) = q(PR) w(E) = w(PR) ret(E) = ret(N) andSyn(E) = andSyn(PR)
24	E → N	**Synteettiset attribuutit** pq(E) = **-[** q(N) **]->** ret(E) = ret(N)
25	E → ..	**Synteettiset attribuutit** q(E) = **<-[** new_v() **]-** ret(E) = new_v()
26	A → *@a_name*	**Synteettiset attribuutit** ret(A) = a_name
27	N → *name*	**Synteettiset attribuutit** q(N) = new_v() **:** *name* name(N) = *name* ret(N) = new_v()
28	N → *	**Synteettiset attribuutit** q(N) = ret(N) = new_v()
29	N → ..	**Synteettiset attribuutit** q(N) = ret(N) = new_v()
30	O → =	**Synteettiset attribuutit** w(O) = **=**
31	O → !=	**Synteettiset attribuutit** w(O) = **<>**
32	O → >	**Synteettiset attribuutit** w(O) = **>**
33	O → <	**Synteettiset attribuutit** w(O) = **<**
34	O → >=	**Synteettiset attribuutit** w(O) = **>=**
35	O → <=	**Synteettiset attribuutit** w(O) = **<=**

36	PR1 → PR2 and PR3	**Periytyneet attribuutit** andInh(PR2) = andInh(PR1) appliesTo(PR2) = appliesTo(PR1) dotInh(PR2) = dotInh(PR1) pathInh(PR2) = pathInh(PR1) transitiveInh(PR2) = transitiveInh(PR1) andInh(PR3) = andSyn(PR2) appliesTo(PR3) = appliesTo(PR1) dotInh(PR3) = dotSyn(PR2) pathInh(PR3) = pathInh(PR1) transitiveInh(PR3) = transitiveInh(PR1) **Synteettiset attribuutit** q(PR1) = q(PR2) q(PR3) w(PR1) = w(PR2) w(PR3) equal(PR1) = equal(PR2) equal(PR3) andSyn(PR1) = andSyn(PR3)
37	PR → PRP	**Periytyneet attribuutit** andInh(PRP) = andInh(PR) **Synteettiset attribuutit** q(PR) = **,** **(** appliesTo(PR) **)** q(PRP) w(PR) = w(PRP) andSyn(PR) = andSyn(PRP) equal(PR) = NULL
38	PR → A O V	**Synteettiset attribuutit** w(PR) = andInh(PR) w(PR) += **ALL(rel in relationships(** pathInh(PR) **) WHERE** , if transitiveInh(PR) w(PR) += **rel** , if transitiveInh(PR) == true w(PR) += appliesTo(PR) else w(PR) += **.** ret(A) w(O) q(V) w(PR) += **)** , if transitiveInh(PR) andSyn(PR) = **AND**
39	PR → A = V	**Synteettiset attribuutit** equal(PR) = dotInh(PR) q(A) **:** q(V) dotSyn(PR) = **,**
40	PR → A	**Synteettiset attribuutit** w(PR) = andInh(PR) w(PR) += **ALL(rel in relationships(** pathInh(PR) **) WHERE** , if transitiveInh(PR) w(PR) += **EXISTS(rel .** ret(A)**)** , if transitiveInh(PR) w(PR) += **EXISTS(**appliesTo(PR) **.** ret(A) **)** else w(PR) += **)** , if transitiveInh(PR) == true

		andSyn(PR) = **AND**
41	PR → SF	**Periytyneet attribuutit** transitiveInh(SF) = transitiveInh(PR) appliesTo(SF) = appliesTo(PR) **Synteettiset attribuutit** w(PR) = andInh(PR) w(SF) andSyn(PR) = **AND**
42	AX → *child*	**Synteettiset attribuutit** start(AX) = **-[** end(AX) = **]->** transitiveSyn(AX) = false
43	AX → *parent*	**Synteettiset attribuutit** start(AX) = **<-[** end(AX) = **]-** transitiveSyn(AX) = false
44	AX → *descendant*	**Synteettiset attribuutit** start(AX) = **-[** end(AX) = ***]->** transitiveSyn(AX) = true
45	AX → *ancestor*	**Synteettiset attribuutit** start(AX) = **<-[** end(AX) = ***]-** transitiveSyn(AX) = true
46	AX → *descendant-or-self*	**Synteettiset attribuutit** start(AX) = **-[** end(AX) ***0..]->** transitiveSyn(AX) = true
47	AX → *descendant-or-self*	**Synteettiset attribuutit** start(AX) = **<-[** end(AX) = ***0..]-** transitiveSyn(AX) = true
48	SF → SFN(A, *"string"*)	**Synteettiset attribuutit** w(SF) = ret(A) w(SF) += **ALL(rel in relationships(** pathInh(SF)**) WHERE** , if transitiveInh(SF) w(SF) += **rel** , if transitiveInh(SF) == true w(SF) += appliesTo(SF) else w(SF) += **.** ret(A) w(SF) **'***string***'** w(SF) += **)** , if transitiveInh(SF) == true andSyn(PR) = **AND**
49	SFN → contains	**Synteettiset attribuutit**

		w(SFN) = **CONTAINS**
50	SFN → starts-`WITH`	**Synteettiset attribuutit** w(SFN) = **STARTS WITH**
51	SFN → ends-`WITH`	**Synteettiset attribuutit** w(SFN) = **ENDS WITH**
52	V → ”*string*”	**Synteettiset attribuutit** q(V) = **‘***string***’**
53	V → *numberOrBool*	**Synteettiset attribuutit** q(V) = *numberOrBool*

Liite 2: Cypher-luontikyselyt

```
MATCH (c1:Customer {contactName: 'Paula Wilson'}), (c2: Customer{contactName:
'Rene Phillips'})
CREATE (c1)-[:FOLLOWS {since: 2010}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Roland Mendel'}), (c2: Customer{contactName:
'Paula Wilson'})
CREATE (c1)-[:FOLLOWS {since: 2005}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Rene Phillips'}), (c2: Customer{contactName:
'Patricia McKenna'})
CREATE (c1)-[:FOLLOWS {since: 2019}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Georg Pipps'}), (c2: Customer{contactName:
'Horst Kloss'}) CREATE (c1)-[:FOLLOWS {since: 2017}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Janete Limeira'}), (c2: Customer{contact-
Name: 'Horst Kloss'})
CREATE (c1)-[:FOLLOWS {since: 2019}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Michael Holz'}), (c2: Customer{contactName:
'Maurizio Moroni'})
CREATE (c1)-[:FOLLOWS {since: 2011}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Jaime Yorres'}), (c2: Customer{contactName:
'Alexander Feuer'})
CREATE (c1)-[:FOLLOWS {since: 2010}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Jaime Yorres'}), (c2: Customer{contactName:
'Marie Bertrand'})
CREATE (c1)-[:FOLLOWS {since: 2012}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Paula Wilson'}), (c2: Customer{contactName:
'Horst Kloss'})
CREATE (c1)-[:FOLLOWS {since: 2018}]->(c2)
```

```
MATCH (c1:Customer {contactName: 'Jaime Yorres'}), (c2: Customer{contactName:
'Horst Kloss'})
CREATE (c1)-[:FOLLOWS {since: 2012}]->(c2)
```

Liite 3. Laajempi jäsennysesimerkki

Tämän liitteen jäsennysesimerkki havainnollistaa laajemmin XPathin ilmaisuvoimaa graafitietokannassa. Jäsennysesimerkkiin sisältyy XPathin ominaisuuksista transitiivinen sulkeuma, sisäkkäiset predikaatit, kaksisuuntainen navigaatio, ja suodatinoperaatiot. Jäsennysesimerkissä haetaan tietokannasta kaikki henkilöt, jotka ovat tehneet tilauksen, tai henkilöt, jotka seuraavat henkilöä, joka on tehnyt tilauksen, tai nämä henkilöt seuraavat henkilöä, joka on tehnyt tilauksen ja niin edelleen. Samalla tilaukseen kuuluu tuotetta nimeltään Tofu vähintään 10 kappaleen verran. Lisäksi tilaukseen kuuluu tuote, jonka valmistajan maa on Tanska. Kuvauksen mukainen XPath-ilmaisu esitetään kyselyssä 5.

```
Customer/descendant-or-self::FOLLOWS/Customer[PURCHASED/Order
[ORDERS[@quantity>=10]/Product/@productName=”Tofu”
and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@coun-
try=”Denmark”]]
```

Kysely 5. Jäsennysesimerkin 2 XPath-kysely

Kysely 5 alkaa hakemalla kaikki Customer-solmut. Kyselyn suhde henkilöstä henkilöön ilmaistaan FOLLOWS-suhteella. Kyselyn kuvauksessa suhde on refleksiivis-transitiivinen, joka voi palauttaa sekä alkuperäisen henkilön, että henkilöön transitiivisten suhteiden kautta liittyvän henkilön. Tämä suhde ilmaistaan lisäämällä XPathin *descendant-or-self*-akseli suhteen nimen eteen. Koska kyselyssä 5 halutaan palauttaa asiakkaat, predikaatin alku asetetaan ennen siirtymistä seuraavaan askeleeseen. Nyt siis refleksiivis-transitiivisella navigaatiolla FOLLOWS-suhteita pitkin saavutetut henkilöt on määritelty kyselyn projektioksi eli kyselyn paluuarvoiksi, sillä XPath-ilmaisu ei jatku predikaatin jälkeen. Predikaatin sisällä kyselyä jatketaan eteenpäin. Ensimmäinen predikaatin sisällä oleva askel on kaariaskel, koska predikaattiin sidottu XPath-askel on solmuaskel. Kaariaskelen solmutesti vaatii suhteen nimeksi PURCHASED-nimisen kaaren, jolla kuvataan asiakkaiden tekemät tilaukset. Tätä suhdetta kuvaavat kaaret on tietokannassa suunnattu Customer-solmuista Order-solmuihin. Tällöin ilmaisussa käytettävä XPathin akseli on *child*. Koska *child*-akseli on XPathissa aina oletusakseli, se on jätetty lisäämättä kyselyyn 5. Kaariaskelta seuraa solmuaskel, jossa solmutestinä on Order. Kyselyn prosessointi jatkuu siis kaikista Order-nimisistä solmuista. Tätä seuraa predikaatti, joka loogisella konjunktiolla ketjuttaa kaksi predikaattia. Ensimmäinen predikaatti alkaa ORDERS-suhteella. Siinä kyselyn kuvauksen sisältämää ehtoa vähintään kymmenen kappaleen määrästä kuvataan attribuutilla *quantity*. Lukumääräsuhde on ilmaistu XPathin operaattorilla >=. Attribuutista on käytetty työhön valittua lyhennettyä muotoa @-notaatiolla. Koska predikaatti on samassa askeleessa ORDERS-solmutestin kanssa, predikaatti on sidottu vain ORDERS-suhteisiin. Polku jatkuu predi-

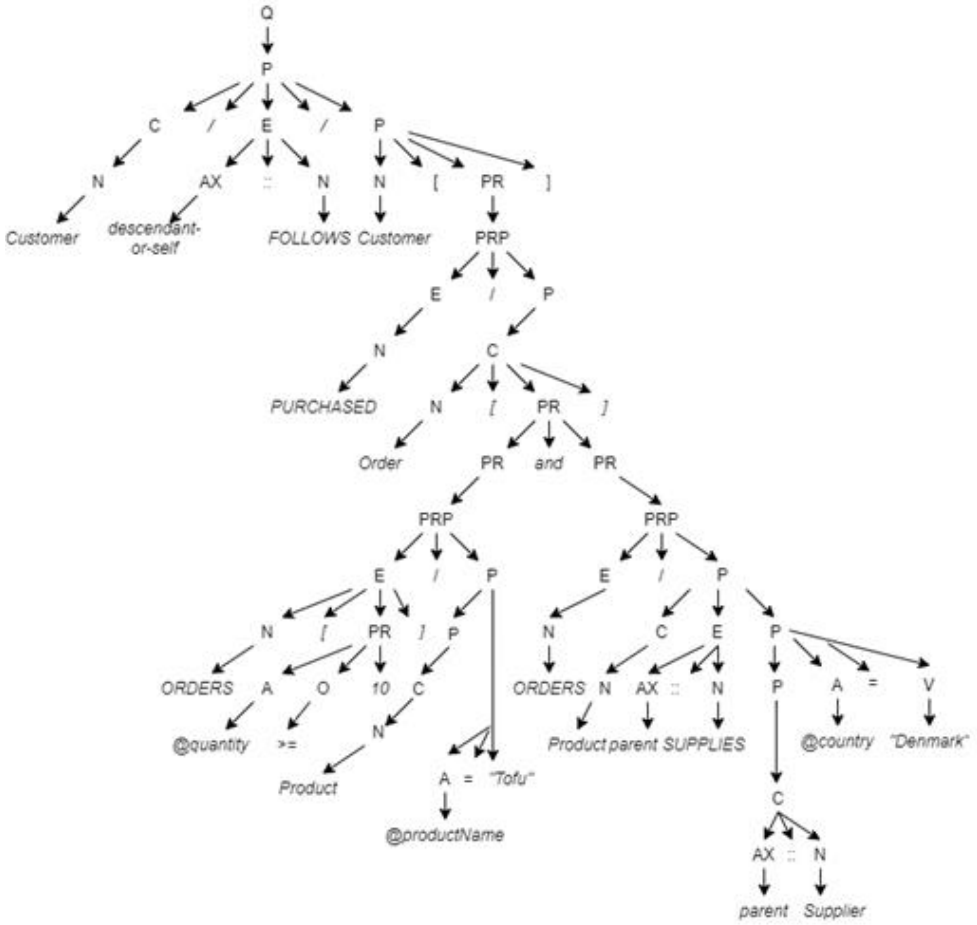
kaatin jälkeen, ja seuraavaksi siirrytään solmuaskeleeseen, jossa solmutestin arvona on Product. Solmutesti ilmaisee, että tilaukseen kuuluu tuote. Product-solmut ilmaisevat tuotetta graafitietokannassa. Lopulta polun päättää predikaatti, joka vastaa kyselyn kuvauksen ehtoa, että tuotteen nimen on oltava Tofu. Tämä ilmaistaan jälleen attribuutin @-notaatiolla ja attribuutin nimellä productName. Merkkijono Tofu on kirjoitettu lainausmerkkien sisälle.

And-operaattoria seuraa toinen predikaatti. Predikaatissa sen ensimmäinen askel tunnistetaan kaariaskeleksi, koska predikaatti on edellisen predikaatin tapaan myös sidottu predikaattia edeltäneeseen solmuaskeleeseen Order. Kyselyn kuvauksen ehto, että tilaukseen kuuluu tuote, kuvataan XPath-kyselyssä siirtymisenä ORDERS-kaarta sen suuntaa pitkin Order-solmusta Product-solmuun. Tuotteen ja valmistajan (Supplier) välinen suhde on kaaviossa kuvattu kaarella, jonka suunta on valmistajasta tuotteeseen. Koska prosessoitavana oleva solmu on Product, suhde kuvataan XPathin *parent*-akselia käyttäen, jolla siis kuvataan siirtyminen kaarta sen käänteiseen suuntaan. Supplier-solmuaskelen jälkeen polussa predikaatilla kuvataan ehto, jonka mukaan tuotteen valmistajan maa on Tanska. Ehto on kuvattu jälleen attribuutilla, XPathin yhtäsuuruusoperaattorilla ja lainausmerkkien ympäröimällä merkkijonolla. Kysely päättyy sulkemalla predikaatit hakasuluilla. Kysely palauttaa graafitietokannasta kyselyyn täsmäävät henkilöt. Kääntämisen ensimmäinen vaihe on tuottaa alkionimien sekvenssi. Kyselyn 5 alkionimien sekvenssi on esitetty taulukossa 12.

tekstialkio	**alkionimi**	tekstialkio	**alkionimi**
Customer	**N**	=	**P**
/	**P**	”Tofu”	**V**
descendant-or-self	**AX**	and	**PR**
::	**E**	ORDERS	**N**
FOLLOWS	**N**	/	**PRP**
/	**P**	Product	**N**
Customer	**N**	/	**PRP**
[	**C**	parent	**AX**
PURCHASED	**N**	::	**E**
/	**PRP**	SUPPLIES	**N**
Order	**N**	/	**PRP**
[	**C**	parent	**AX**
ORDERS	**N**	::	**C**
[	**E**	Supplier	**N**
@quantity	**A**	/	**P**
>=	**O**	@country	**A**
10	**V**	=	**P**
/	**PRP**	”Denmark”	**V**
Product	**N**	]	**C**
/	**P**	]	**C**
@productName	**A**		

Taulukko 12. Alkionimien sekvenssi

Kuvassa 8 on esitetty alkionimien sekvenssistä muodostettu jäsennyspuu. Kuvan 8 jäsennyspuussa terminaalisymbolit ovat jälleen jäsennyspuun lehtisolmuja ja esitetty puussa kursivoituna.



Kuva 8. Jäsennysesimerkin 2 jäsennyspuu

Kolmannessa vaiheessa eli semanttisella tasolla, jäsennyspuuta käydään läpi, ja muodostetaan semanttisten sääntöjen mukaan semantiikka ilmaisulle. Tämän jäsennysesimerkin sääntöjä luetaan siten, että taulukon vasemmanpuoleisimman sarakkeen ensimmäinen luku kertoo järjestyksen, milloin jäsennyspuun symboliin liittyvään sääntöön saavutaan syvyyshaun mukaisessa järjestyksessä. Tämä luku on jokaisella säännöllä yksilöivä. Toinen luku viittaa liitteen 1 mukaisen kieliopin sääntöön. Semanttisissa säännöissä attribuuttikielioppiin kuuluvat periytyneet ja synteettiset attribuutit on eroteltu toisistaan. Lisäksi jokaisen periytyneen attribuutin yhteydessä on kursivoituna sulkujen sisällä numero, joka viittaa jäsennysesimerkkiin kuuluvien sääntöjen yksilölliseen numeroon, ja kertoo mille säännölle attribuutin arvo periytyy. Tämä linkittää säännöt toisiinsa ja tekee semanttisten sääntöjen luomisen seuraamisesta hieman helpompaa.

Customer/descendant-or-self::FOLLOWS/Customer[PURCHASED/Order[ORDERS[@quantity>=10]/Product/@productName=”Tofu”
and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]]

1 1	Q → P	**Periytyneet attribuutit** andInh(P) = NULL *(2)* appliesTo(P) = NULL *(2)* finishPath(P) = false *(2)* insidePredicate(P) = false *(2)* pathInh(P) = NULL *(2)* **Synteettiset attribuutit** ret(Q) = `MATCH` (a9:Customer), p0=(a9)-[:FOLLOWS *0..]->(a7:Customer), (a7)-[a4: PURCHASED]-> (a6:Order), (a6)-[a5:ORDERS]->(a4:Product), (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) `WHERE` a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ `RETURN` a7

Customer/descendant-or-self::FOLLOWS/Customer[PURCHASED/Order
[ORDERS[@quantity>=10]/Product/@productName=”Tofu”
and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]]

2 6	P1 → C / E / P2	**Periytyneet attribuutit** andInh(C) = NULL *(3)* finishPath(C) = false *(3)* insidePredicate(C) = false *(3)* andInh(E) = NULL *(5)* appliesTo(E) = a9 *(5)* insidePredicate(E) = false *(5)* andInh(P2) = NULL *(8)* finishPath(P2) = false *(8)* transitiveInh(P2) = true *(8)* **Synteettiset attribuutit** pq(P1) = (a9:Customer), p0=(a9)-[:FOLLOWS *0..]->(a7:Customer) q(P1) = , (a7)-[a4: PURCHASED]->(a6:Order), (a6)-[a5:ORDERS]->(a4:Product), (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(P1) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ ret(P1) = a7 andSyn(P1) = AND

Customer

3 20	C → N	**Synteettiset attribuutit** pq(C) = (a9:Customer) ret(C) = a9

Customer

4 27	N → *name*	**Synteettiset attribuutit** q(N) = a9:Customer name(N) = Customer ret(N) = a9

descendant-or-self::FOLLOWS

5 22	E → AX::N	**Synteettiset attribuutit** pq(E) = -[:FOLLOWS *0..]->
ret(E) = a8 pathSyn(E) = p0 pathSeq(E) = , p0=(a9)

descendant-or-self

6 46	AX → *descendant-or-self*	**Synteettiset attribuutit** start(AX) = -[end(AX) = *0..]-> transitiveSyn(AX) = true

FOLLOWS

7 27	N → *name*	**Synteettiset attribuutit** q(N) = a8:FOLLOWS name(N) = FOLLOWS ret(N) = a8

Customer[PURCHASED/Order[ORDERS[@quantity>=10]/Product/@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]]

8 8	P → C	**Periytyneet attribuutit** andInh(C) = NULL *(9)* pathInh(C) = NULL *(9)* **Synteettiset attribuutit** pq(P) = (a7:Customer) q(P) = , (a7)-[a4: PURCHASED]-> (a6:Order), (a6)-[a5: ORDERS]->(a4: Product), (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(P) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ ret(P) = a7 andSyn(P) = AND pathSyn(P) = NULL

Customer[PURCHASED/Order[ORDERS[@quantity>=10]/Product@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]]

9 19	C → N[PR]	**Periytyneet attribuutit** andInh(PR) = NULL *(11)* dotInh(PR) = NULL *(11)* appliesTo(PR) = a7 *(11)* **Synteettiset attribuutit** pq(C) = (a7:Customer) q(C) = , (a7)-[a4: PURCHASED]-> (a6:Order), (a6)-[a5: ORDERS]->(a4: Product), (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(C) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ ret(C) = a7 andSyn(C) = AND

Customer

10 27	N → *name*	**Synteettiset attribuutit** q(N) = a7:Customer name(N) = Customer ret(N) = a7

PURCHASED/Order[ORDERS[@quantity>=10]/Product/@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]

11 37	PR → PRP	**Periytyneet attribuutit** andInh(PRP) = NULL *(12)* **Synteettiset attribuutit** q(PR) =, (a7) -[a4: PURCHASED]->(a6:Order), (a6)-[a5: ORDERS]->(a4: Product), (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(PR) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ andSyn(PR) = AND equal(PR) = NULL

PURCHASED/Order[ORDERS[@quantity>=10]/Product/@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]

12 16	PRP → E / P	**Periytyneet attribuutit** andInh(E) = NULL *(13)* pathInh(E) = NULL *(13)* insidePredicate(E) = true *(13)* andInh(P) = NULL *(15)* insidePredicate(P) = true *(15)* appliesTo(P) = a4 *(15)* transitiveInh(P) = false *(15)* **Synteettiset attribuutit** q(P1) = -[a4: PURCHASED]-> (a6:Order), (a6)-[a5: ORDERS]->(a4: Product), (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(P1) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ pathSyn(PRP) = NULL andSyn(PRP) = AND

PURCHASED

13 24	E → N	**Synteettiset attribuutit** pq(E) = -[a4: PURCHASED]-> ret(E) = a4

PURCHASED

14 27	N → *name*	**Synteettiset attribuutit** q(N) = a4: PURCHASED name(N) = PURCHASED ret(N) = a4

Order[ORDERS[@quantity>=10]/Product/@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]]

15 8	P → C	**Periytyneet attribuutit** andInh(C) = NULL *(16)* **Synteettiset attribuutit** pq(P) = (a6:Order) q(P) = , (a6)-[a5: ORDERS]->(a4: Product) , (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(P) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ ret(P) = a6 pathSyn(P) = NULL andSyn(P) = NULL

Order[ORDERS[@quantity>=10]/Product/@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”]]

16 19	C → N[PR]	**Periytyneet attribuutit** appliesTo(PR) = a6 *(18)* dotInh(PR) = NULL *(18)* andInh(PR) = NULL *(18)* **Synteettiset attribuutit** pq(C) = (a6:Order) q(C) = , (a6)-[a5: ORDERS]->(a4: Product) , (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(C) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ ret(C) = a6 andSyn(C) = AND

Order

17 27	N → *name*	**Synteettiset attribuutit** q(N) = a6:Order name(N) = Order ret(N) = a6

ORDERS[@quantity>=10]/Product/@productName=”Tofu” and ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”

18 36	PR1 → PR2 and PR3	**Periytyneet attribuutit** appliesTo(PR2, PR3) = a6 *(19, 33)* pathInh(PR2, PR3) = NULL *(19, 33)* transitiveInh(PR2, PR3) = false *(19, 33)* andInh(PR2) = NULL *(19)* andInh(PR3) = AND *(33)* **Synteettiset attribuutit** q(PR1)= , (a6)-[a5: ORDERS]->(a4: Product) , (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(PR1) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ AND a0.country = ‘Denmark’ andSyn(PR1) = AND equal(PR1) = NULL

ORDERS[@quantity>=10]/Product/@productName=”Tofu”

19 37	PR → PRP	**Periytyneet attribuutit** andInh(PRP) = NULL *(20)* **Synteettiset attribuutit** q(PR) =, (a6)-[a5: ORDERS]->(a4: Product) w(PR) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ andSyn(PR) = AND equal(PR) = NULL

ORDERS[@quantity>=10]/Product/@productName=”Tofu”

20 16	PRP → E / P	**Periytyneet attribuutit** pathInh(E) = NULL *(21)* andInh(E) = NULL *(21)* pathInh(P) = NULL *(27)* andInh(P) = AND *(27)* **Synteettiset attribuutit** q(PRP) = -[a5: ORDERS]->(a4: Product}) w(PRP) = a5.quantity >= 10 AND a4.productName = ‘Tofu’ pathSyn(PRP) = NULL andSyn(PRP) = NULL

ORDERS[@quantity>=10]

21 23	E → N[PR]	**Periytyneet attribuutit** appliesTo(PR) = a5 *(23)* andInh(PR) = NULL *(23)* transitiveInh(PR) = false *(23)* **Synteettiset attribuutit** pq(E) = -[a5: ORDERS]-> q(E) = NULL w(E) = a5.quantity >= 10 ret(E) = a6 andSyn(E) = AND

ORDERS

22 27	N → *name*	**Synteettiset attribuutit** q(N) = a5: ORDERS name(N) = ORDERS ret(N) = a5

@quantity>=10

23 38	PR → A O V	**Synteettiset attribuutit** w(PR) = a5.quantity >= 10 andSyn(PR) = AND

@quantity

24 26	A → *@a_name*	**Synteettiset attribuutit** ret(A) = quantity

>=

25 34	O → >=	**Synteettiset attribuutit** w(O) = >=

10

26 53	V → *numberOrBool*	**Synteettiset attribuutit** q(V) = 10

Product/@productName=”Tofu”

27 11	P1 → P2 / A = V	**Periytyneet attribuutit** pathInh(P2) = NULL *(28)* andInh(P2) = AND *(28)* transitiveInh(P2) = false *(28)* insidePredicate(P2) = true *(28)* **Synteettiset attribuutit** pq(P) = (a4: Product) q(P) = NULL w(P) = AND a4.productName = ‘Tofu’ ret(P) = a4 pathSyn(P) = NULL

Product

28 8	P → C	**Periytyneet attribuutit** appliesTo(PR) = a4 *(30)* dotInh(PR) = NULL *(30)* andInh(PR) = AND *(30)* **Synteettiset attribuutit** pq(C) = (a4: Product) q(C) = NULL w(C) = NULL ret(C) = a4 andSyn(C) = NULL

Product

29 20	C → N	**Synteettiset attribuutit** pq(N) = (a4: Product) ret(N) = a4

Product

30 27	N → *name*	**Synteettiset attribuutit** q(N) = a4: Product name(N) = Product ret(N) = a4

@productName

31 26	A → *@a_name*	**Synteettiset attribuutit** ret(A) = productName

“Tofu”

32 52	V → ”*string*”	**Synteettiset attribuutit** q(V) = ‘Tofu’

ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”

33 37	PR → PRP	**Periytyneet attribuutit** andInh(PRP) = AND *(34)* **Synteettiset attribuutit** q(PR) = (a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(PR) = AND a0.country = ‘Denmark’ andSyn(PR) = AND equal(PR) = NULL

ORDERS/Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”

34 16	PRP → E / P	**Periytyneet attribuutit** andInh(E) = AND *(35)* insidePredicate(E) = true *(35)* insidePredicate(P) = true *(37)* andInh(P) = AND *(37)* appliesTo(P) = a3 *(37)* transitiveInh(P) = false *(37)* **Synteettiset attribuutit** q(PRP) = -[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) w(PRP) = AND a0.country = ‘Denmark’ pathSyn(PRP) = NULL andSyn(PRP) = AND

ORDERS

35 24	E → N	**Synteettiset attribuutit** pq(E) = -[a3:ORDERS]-> ret(E) = a3

ORDERS

36 27	N → *name*	**Synteettiset attribuutit** q(N) = a3:ORDERS name(N) = ORDERS ret(N) = a3

Product/parent::SUPPLIES/parent::Supplier/@country=”Denmark”

37 6	P1 → C / E / P2	**Periytyneet attribuutit** andInh(C) = AND *(38)* finishPath(C) = false *(38)* insidePredicate(C) = false *(38)* andInh(E) = AND *(40)* appliesTo(E) = a2 *(40)* insidePredicate(E) = true *(40)* andInh(P2) = AND *(43)* finishPath(P2) = false (43*)* insidePredicate(P2) = false *(43)* pathInh(P2) = NULL *(43)* transitiveInh(P2) = false *(43)* appliesTo(P2) = a1 *(43)* **Synteettiset attribuutit** pq(P1) = (a2:Product)<-[a1:SUPPLIES]-(a0: Supplier) q(P1) = NULL w(P1) = AND ret(P1) = a0 andSyn(P1) = AND

Product

38 20	C → N	**Synteettiset attribuutit** pq(C) = (a2: Product) ret(C) = a2

Product

39 27	N → *name*	**Synteettiset attribuutit** q(N) = a2:Product name(N) = Product ret(N) = a2

parent::SUPPLIES

40 22	E → AX::N	**Synteettiset attribuutit** pq(E) =<-[a1: SUPPLIES]- pathSyn(E) = p0 ret(E) = a1

parent

41 43	AX → *parent*	**Synteettiset attribuutit** start(AX) = <-[end(AX) =]- transitive(AX) = false

SUPPLIES

42 27	N → *name*	**Synteettiset attribuutit** q(N) = a1:SUPPLIES name(N) = SUPPLIES ret(N) = a1

parent::Supplier/@country=”Denmark”

43 11	P1 → P2/A=V	**Periytyneet attribuutit** andInh(P2) = AND *(44)* insidePredicate(P2) = true *(44)* transitiveInh(P2) = false *(44)* **Synteettiset attribuutit** pq(C) = (a0: Supplier) q(C) = NULL w(C) = AND a0.country = ‘Denmark’ ret(C) = a0 andSyn(C) = AND

parent::Supplier

44 8	P → C	**Periytyneet attribuutit** andInh(C) = AND *(45)* pathInh(C) = NULL *(45)* insidePredicate(C) = true *(45)* **Synteettiset attribuutit** pq(P) = (a0: Supplier) q(P) = NULL w(P) = NULL ret(P) = a0 andSyn(P) = AND

parent::Supplier

45 18	C → AX::N	**Synteettiset attribuutit** pq(C) = (a0) ret(C) = a0

parent

46 39	AX → *parent*	**Synteettiset attribuutit** start(AX) = <-[end(AX) =]- transitive(AX) = false

Supplier

47 23	N → *name*	**Synteettiset attribuutit** q(N) = a0:Supplier name(N) = Supplier ret(N) = a0

@country

48 22	A → *@a_name*	**Synteettiset attribuutit** ret(A) = country

"Denmark"

49 48	V → "*string*"	**Synteettiset attribuutit** q(V) = 'Denmark'

Esimerkistä 2 muodostunut Cypher-kysely on jäsennyspuun juurisolmun eli säännön 1 palauttama merkkijono on ilmaistu kyselyssä 6.

```
MATCH  (a9:Customer),p0=(a9)-[:FOLLOWS *0..]->(a7:Customer),  (a7)-
[a4:PURCHASED]->(a6:Order),
(a6)-[a5:ORDERS]->(a4:Product),
(a6)-[a3:ORDERS]->(a2:Product)<-[a1:SUPPLIES ]-(a0:Supplier
WHERE a5.quantity >= 10 AND a4.productName = 'Tofu' AND a0.country
= 'Denmark'
RETURN a7
```

Kysely 6. Laajemman jäsennysesimerkin XPath-ilmaisun käännös Cypher-kielen ilmaisuna

Liite 4. Esimerkkikyselyitä

Kysely 7.

Kuvaus	Hakee kaikki asiakkaat
XPath	`Customer`
Cypher	`MATCH (a0:Customer)` `RETURN a0`

Kysely 8.

Kuvaus	Hakee kaikkien tilausten rahtihinnan keskiarvon (aggregaatti)
XPath	`avg(Order/@freight)`
Cypher	`MATCH (a0:Order)` `RETURN avg(a0.freight)`

Kysely 9.

Kuvaus	Kysely hakee kaikki ne tilaukset, jonka on tehnyt asiakas, joka asuu Lontoossa ja tilaus kuuluu kategoriaan, jonka nimi on Food (konjunktiivinen kysely)
XPath	`Order[parent::PURCHASED/Customer/@city="London" and PART_OF/Category/@name="Food"]`
Cypher	`MATCH (a0:Order), (a0)<-[a1:PURCHASED]-(a2:Customer {city: 'London'}), (a0)-[a3:PART_OF]->(a4:Category {name: 'Food'})` `RETURN a0`

Kysely 10.

Kuvaus	Hakee kaikki henkilöt, joiden kaverien kaverit ovat tehneet tilauksen Tofu-nimisestä tuotteesta (RPQ)
XPath	`Customer[descendant::FOLLOWS/Customer/ PURCHASED/Order/ORDERS/Product[@name="Tofu"]]`
Cypher	`MATCH (a0:Customer), (a0)-[:FOLLOWS*]->(a1:Customer)-[a2:PUR-CHASED]->(a4:Order)-[a5:ORDERS]->(a6:Product {name: 'Tofu'}) RETURN a0`

Kysely 11.

Kuvaus	Haetaan Tofu-nimisestä tuotteesta tilauksen tehneiden asiakkaiden seuraajat ja seuraajien seuraajat ja niin edelleen, missä seuraajat ovat niin ikään asiakkaita (Kaksisuuntainen navigaatio)
XPath	`Customer/ancestor::FOLLOWS/Customer`
Cypher	`MATCH (a0:Customer)<-[:FOLLOWS*]-(a1:Customer) RETURN a1`

Kysely 12a.

Kuvaus	Hakee kaikki asiakkaiden väliset transitiiviset seuraamissuhteet, joissa seuraamissuhteen alkamista kuvaava ominaisuus since on yhtä kuin 2010. (RPQ vertailuoperaatioiden kanssa)
XPath	`Customer/descendant::FOLLOWS[@since=2010]/Customer`
Cypher	`MATCH (a0:Customer)-[:FOLLOWS {since: 2010}*]->(a1: Customer) RETURN a1`

Kysely 12b.

Kuvaus	Hakee kaikki asiakkaiden väliset transitiiviset seuraamissuhteet, joissa seuraamissuhde on ollut voimassa aikaisintaan vuodesta 2010
XPath	`Customer/descendant::FOLLOWS[@since>=2010]/Customer`
Cypher	`MATCH p0=(a0:Customer)-[:FOLLOWS*]->(a1: Customer) WHERE ALL(rel in relationships(p0)) WHERE rel.since >= 2010 RETURN a1`