Active XML

Tamara VOS

promotor :
Prof. dr. Jan VAN DEN BUSSCHE

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica afstudeerrichting databases

universiteit hasselt

Woord vooraf

Het feit dat Active XML volledig gebaseerd is op standaarden, doet vermoeden dat deze documenten en systemen in de toekomst meer en meer gebruikt zullen worden. De eerste papers hieromtrent werden geschreven in 2003, wat de actualiteit van dit onderwerp in de verf zet.[1] *Dit alles gaf me de overtuiging dat een thesis over dit onderwerp, een leerrijke ervaring zou worden.*

Het doel van deze thesis, was samen met mijn promotor onderzoek te voeren over Active XML. De exacte inhoud van de thesis lag in eerste instantie niet vast. Gaandeweg werd het te volgen pad duidelijk. Door deze samenwerking kon ik veel meer bereiken als bij een meer individueel werk. Ik wil dan ook mijn promotor Professor Jan Van den Bussche bedanken voor de vlotte samenwerking. Elke afspraak voor mijn thesis leidde tot een beter begrip van de stof en gaf me een positieve stimulans om verder te werken.

Verder wil ik ook mijn broer bedanken voor het nalezen van mijn thesis. En tenslotte nog een woordje van dank aan mijn ouders voor de kansen die zij mij gegeven hebben.

Tamara Vos
Diepenbeek, juni 2006

[1] Hiermee wil ik niet zeggen dat papers uit vroegere periodes minder van belang zijn. Zo heb ik voor mijn thesis een paper uit 1983 geconsulteerd. Dat is geschreven voor ik geboren werd!

Inhoudsopgave

Hoofdstuk 1

Inleiding tot Active XML

In dit eerste hoofdstuk brengen we alle belangrijke aspecten van Active XML, afgekort AXML, aan het licht, om een duidelijk beeld te kunnen geven van het probleemdomein. Als slot van dit hoofdstuk komen we tot de probleemstelling van deze thesis.

1.1 XML en web services

XML of Extensible Mark up Language (W3C, 1996) is het standaard formaat voor de uitwisseling van informatie via het Internet (Abiteboul *et al.*, 2004a). Het XML data model is zelf beschrijvend in die zin dat niet enkel de data zelf maar ook een indicatie naar de betekenis ervan opgeslagen wordt. Logisch gezien kunnen we een XML document als een gelabelde en geordende boom bekijken. Hoewel zulke bomen in principe elke vorm kunnen aannemen, heeft de auteur door middel van bijvoorbeeld XML Schema's (W3C, 2001b) de mogelijkheid om XML documenten een bepaalde structuur op te leggen.

Web services, die een stijgende populariteit kennen, zijn applicaties die uitgevoerd kunnen worden over het web. De Amazon web services zijn voorbeelden hiervan (Amazon.com, 2003). Een aantal standaarden maken web services, net als XML, wereldwijd toegankelijk. De UDDI directory (Universal Discovery Description and Integration (OASIS, 2000)) maakt het mogelijk om een bepaalde service te vinden. WSDL (Web Service Definition Language (W3C, 2001a)) beschrijft de manier waarop men met een web service kan interageren. Tenslotte kan de service aangeroepen worden, gebruikmakend van SOAP (Simple Object Access Protocol (W3C, 2004)).

1.2 Uitbreiding naar AXML en AXML web services

Het succes van XML en web services heeft geleid tot de introductie van een nieuwe generatie XML documenten, waarin een deel van de data expliciet gegeven is en een deel enkel intentioneel, door middel van calls naar web services, ingebed in de documenten. Dit soort

XML documenten noemt men *Active XML documenten*. Active XML kan afgekort worden tot AXML. In deze thesis gebruik ik beide begrippen door elkaar. Door het toevoegen van web services zijn de AXML documenten niet langer statisch, gezien het uitvoeren van services nieuwe, up-to-date informatie toevoegt.
Ook het concept van web services werd uitgebreid. Web services die AXML data uitwisselen, door het gebruik van AXML documenten als parameters en resultaten, worden overeenkomstig AXML web services genoemd.
AXML systemen zijn bedoeld voor data management op Internet, in een peer-to-peer architectuur. Elke peer houdt een aantal AXML documenten bij en kan deze documenten verrijken met nieuwe data door de service calls te activeren. Een service call die verschillende keren aangeroepen wordt, kan telkens een ander antwoord teruggeven. Daarom zijn AXML documenten dynamisch. Een peer levert ook een aantal web services aan de andere, die gedefinieerd worden als queries of updates over zijn verzameling AXML documenten. Op die manier heeft elke peer zowel een client- als een server-rol.

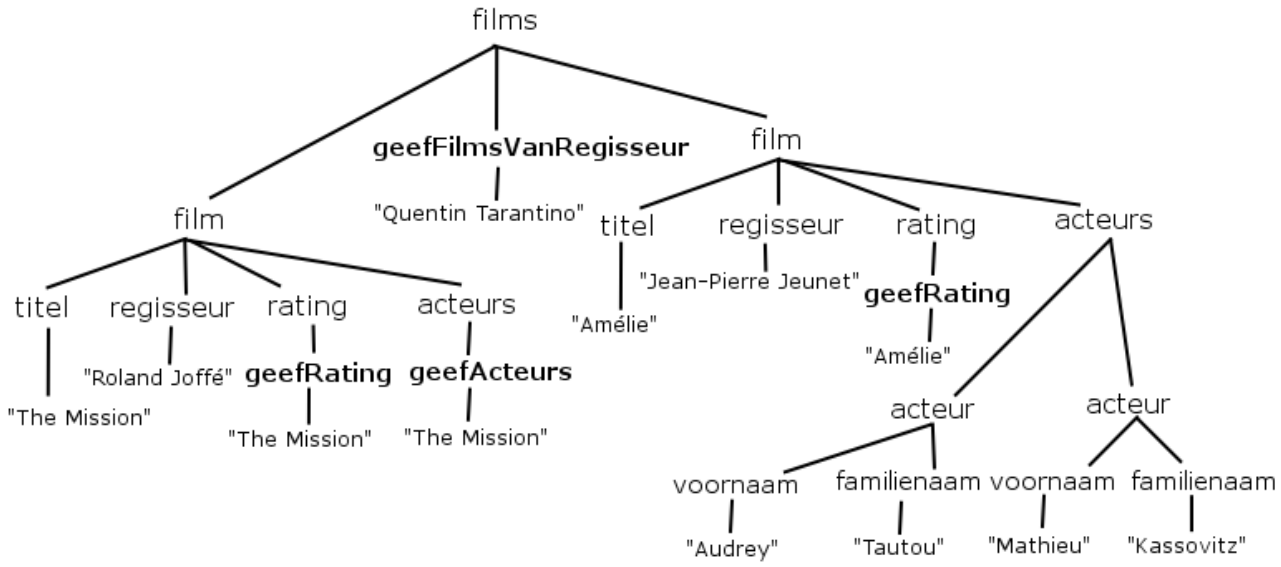
AXML web services geven actieve antwoorden als resultaat. In Abiteboul *et al.* (2004b) worden logische en fysische motivaties gegeven voor het beantwoorden van queries met actieve antwoorden. De logische motivaties stellen dat actieve antwoorden meer functionaliteit leveren. Hieronder worden een aantal motivaties geschetst.
Een van de belangrijkste motivaties is het feit dat de informatie deels afhankelijk kan zijn van de tijd. Wanneer bijvoorbeeld informatie over een skigebied opgevraagd wordt, zal de huidige sneeuwconditie vermeld worden. Door dergelijke data actief te bewaren kan men de informatie steeds updaten naar de huidige omstandigheden. Een ander gevolg is dat men op de hoogte is van die web service en deze met andere gelijkaardige parameters kan aanroepen. Dit maakt het bijvoorbeeld mogelijk om de sneeuwcondities van andere skigebieden op te vragen.
Nu geven we een aantal praktische toepassingen. Soms wil men een onvolledig antwoord verkrijgen. Wanneer we de query „active xml" ingeven in Google bekomen we circa 514.000 resultaten. De server zal niet alle resultaten doorsturen, maar enkel de eerste tien en een link naar de andere resultaten. Om een geheim telefoonnummer te versturen, wordt het geëncrypteerd vóór het zenden. De gebruiker moet het dus eerst decrypteren om de informatie te begrijpen. Deze twee toepassingen zijn ook actieve antwoorden.
Enkele fysische motivaties leiden tot het schema dat zenders en ontvangers van AXML data moeten overeenkomen. Dit wordt besproken in sectie 1.4.2.

1.3 Voorbeeld AXML document

Figuur 1.1 geeft een voorbeeld van een Active XML document, samen met diens logische boomvoorstelling. Het document bevat informatie over films: titel, regisseur, rating en de

voornaamste acteurs die meespelen. De syntax van het document is analoog aan de voorbeelden uit Abiteboul *et al.* (2004b). De service calls zitten in `sc`-elementen, waarbij de naam van de web service aangegeven wordt. Uiteraard zal in de praktijk voor services meer gespecificeerd moeten worden, zoals de url van de server, de manier waarop de connectie dient te gebeuren,... We hanteren de conventie om in de boomvoorstelling service calls vetgedrukt weer te geven en datawaardes tussen dubbele aanhalingstekens te plaatsen. De knopen gelabeld met namen van web services noemen we functieknopen.

```
<films>
  <film>
    <titel>"The Mission"</titel>
    <regisseur>"Roland Joffé"</regisseur>
    <rating><sc service="geefRating">"The Mission"</sc></rating>
    <acteurs><sc service="geefActeurs">"The Mission"</sc></acteurs>
  </film>
  <sc service="geefFilmsVanRegisseur">"Quentin Tarantino"</sc>
  <film>
    <titel>"Amélie"</titel>
    <regisseur>"Jean-Pierre Jeunet"</regisseur>
    <rating><sc service="geefRating">"Amélie"</sc></rating>
    <acteurs>
      <acteur><voornaam>"Audrey"</voornaam><achternaam>"Tautou"</achternaam></acteur>
      <acteur><voornaam>"Mathieu"</voornaam><achternaam>"Kassovitz"</achternaam></acteur>
    </acteurs>
  </film>
</films>
```



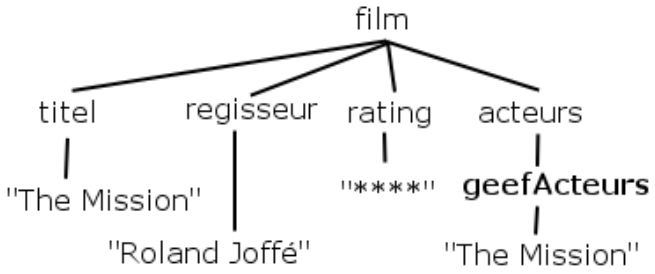
Figuur 1.1: Voorbeeld AXML document en bijhorende boomvoorstelling

1.4 Web services in AXML documenten

1.4.1 Uitvoering service call

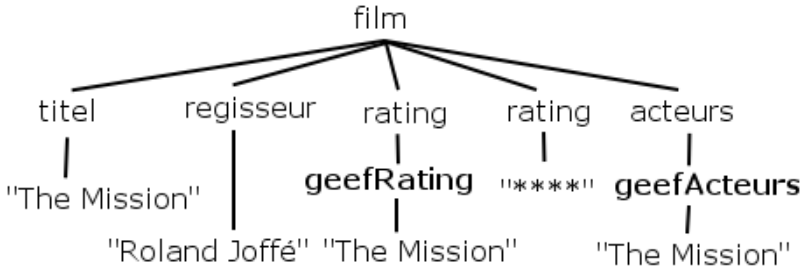
Een AXML document bevat functieknopen, die overeenkomen met een bepaalde web service. De knoop bevat de nodige data om de service uit te voeren, waarbij de kinderen van de knoop als parameters meegegeven worden. Het toevoegen van het resultaat van een service call aan een AXML document noemen we het materialiseren van de data van de service call. Dit kan op verschillende manieren gebeuren.

1. In Abiteboul *et al.* (2004a) wordt een functieknoop vervangen door het resultaat van de service. In figuur 1.2 ziet u een deel van het voorbeelddocument (figuur 1.1), waarin het resultaat van de `geefRating` service ingevoegd is en dus de functieknoop en zijn enige kindknoop vervangt.



Figuur 1.2: Resultaat `geefRating` functie vervangt de functieknoop

2. Daarentegen wordt in Abiteboul *et al.* (2004c,b) de return waarde toegevoegd als sibling van de functieknoop. In figuur 1.3 ziet u opnieuw het resultaat van de `geefRating` functie voor deze tweede manier. Bij deze strategie mag een service call enkel uitgevoerd worden indien daardoor het AXML document gewijzigd wordt, meer bepaald zal de service een document moeten verrijken met nieuwe informatie.



Figuur 1.3: Resultaat `geefRating` functie als sibling toegevoegd

3. In Active XML team (2003a) worden nog andere mogelijkheden aangegeven, waarbij bestaande elementen uit het AXML document samengevoegd worden met nieuwe elementen, op basis van hun id of key specificaties.

1.4.2 Materialisatie service calls

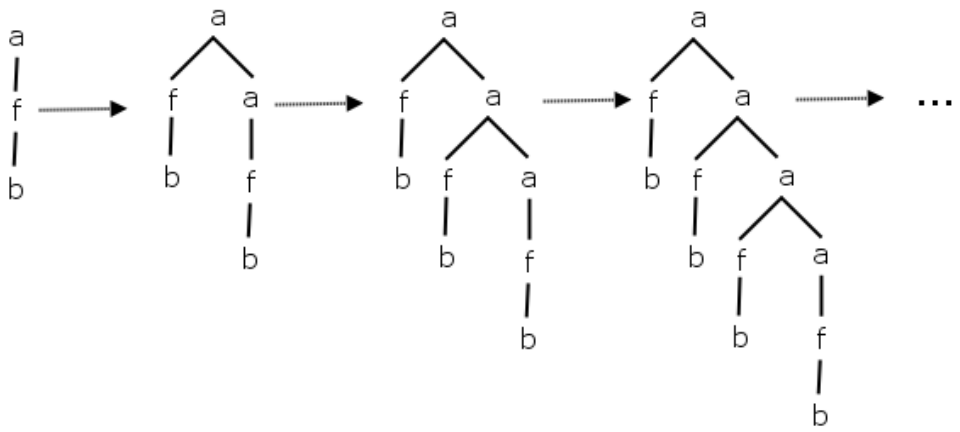
In Active XML documenten wordt de data deels intentioneel opgeslagen. Dat heeft als voordeel dat, op het moment van de activatie van de service, up-to-date informatie bekomen wordt. Het moment waarop data van service calls gematerialiseerd moet worden, is niet voor elk document of voor elke service gelijk. De activatie van service calls kan op geregelde tijdstippen gebeuren of het kan afhankelijk zijn van bepaalde gebeurtenissen. Zoals later nog besproken wordt, kan een service ook pas uitgevoerd worden als het resultaat ervan nodig is voor het uitvoeren van een query over een AXML document. In elk van deze gevallen regelt de client het uitvoeren van de services. Dit noemt men *pull modus*. AXML biedt ook de mogelijkheid om service calls te activeren langs de kant van de server, *push modus* genaamd. In dat geval wordt de service eenmaal geactiveerd en gelinkt aan documenten. De server houdt de documenten continu up-to-date, door alle nieuwe informatie te pushen.

AXML wordt gebruikt voor het uitwisselen van informatie over Internet. Zender en ontvanger moeten een bepaald schema overeenkomen waaraan alle verstuurde documenten moeten voldoen. Aan de hand daarvan weet de zender precies welke services hij moet activeren en welke niet. De keuze van het schema dient te gebeuren aan de hand van verschillende factoren zoals performantie, communicatiekost of veiligheidsredenen. Stel dat de zender van een AXML document overbelast is, dan zal hij het materialiseren zoveel mogelijk overlaten aan de ontvanger, om de transactie zo performant mogelijk te laten verlopen. Het kan echter ook noodzakelijk zijn dat de zender bepaalde service calls uitvoert, omdat de ontvanger niet over de rechten beschikt om het zelf te doen. Wanneer er echter geen problemen zijn met rechten, kan men een lagere communicatiekost bekomen door de calls niet uit te voeren voor de verzending.

1.4.3 Recursieve service calls

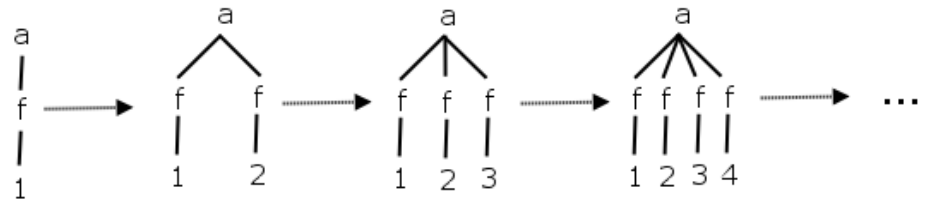
Web services kunnen op twee manieren voor recursie zorgen. Enerzijds kan een service call zelf een call naar een andere service activeren. Anderzijds kan het resultaat van een service nieuwe service calls bevatten. Daarom zullen sommige materialisaties nooit eindigen.
Het spreekt voor zich dat recursie kan leiden tot AXML documenten die oneindig zijn in de lengte. Als we de functieknoop vervangen door het resultaat van de functie, is dit ook de enige mogelijkheid. Echter in het model van Abiteboul *et al.* (2004c) waarbij het resultaat van een service als sibling van de functieknoop toegevoegd wordt, kan een oneindig AXML document, ook oneindig zijn in de breedte.

Beschouw bijvoorbeeld `<a><sc service="f"> b </sc></a>`. Veronderstel dat f een service is die op input b, volgende AXML data teruggeeft: `<a><sc service="f"> b </sc></a>`. Bij herhaalde activaties van de service bekomen we een AXML document dat oneindig is in de lengte, zoals geïllustreerd in figuur 1.4.



Figuur 1.4: Herhaalde activatie van service f leidt tot een AXML document oneindig in de lengte

Een document oneindig in de breedte bekomen we bijvoorbeeld door een service f, die op input van een natuurlijk getal n, opnieuw een functie element f oplevert met parameter $n+1$. Dit zien we in figuur 1.5.

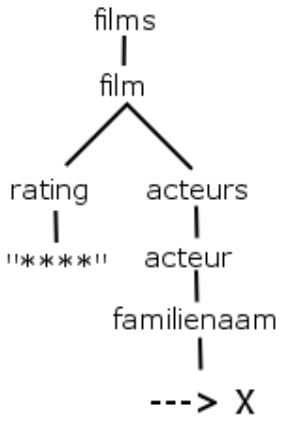


Figuur 1.5: Herhaalde activatie van service f leidt tot een AXML document oneindig in de breedte

1.5 Queries over AXML documenten

Zoals de XML querytalen, XQuery (W3C, 2005) en XPath (W3C, 1999), de mogelijkheid bieden om informatie te zoeken in traditionele XML documenten, willen we ook AXML documenten kunnen ondervragen. Abiteboul *et al.* (2004a) stelt queries over AXML documenten voor als boompatronen. Dit zijn gelabelde bomen waarin de labels bestaan uit variabele namen, constanten (namen van elementen of constante waardes) of het speciale symbool *. Boompatroonqueries houden geen rekening met de volgorde van elementen in AXML documenten. Ook worden enkel de dataknopen uit het document bekeken. De functieknopen zijn

een middel om aan de documenten nieuwe informatie toe te voegen. Pas wanneer dat gebeurd is, kan die informatie in het resultaat van queries voorkomen.
In figuur 1.6 ziet u een voorbeeld van een boompatroonquery, gemaakt aan de hand van het voorbeeld in Abiteboul *et al.* (2004a). De query zoekt de familienamen van de hoofdacteurs uit films met vier sterren. De waardes op de plaats van de X (waar het pijltje naar wijst), zitten in het resultaat van de query.



Figuur 1.6: Voorbeeld boompatroonquery

De query kan ook als volgt weergegeven worden:

```
familienamen{x} :- films{film{rating{"****"},
                              acteurs{acteur{familienaam{x}}}}}.
```

In sectie 1.6 wordt deze syntax wat meer uitgelegd.

1.5.1 Query evaluatie

Voor het evalueren van queries zijn globaal gezien twee strategieën (Abiteboul *et al.*, 2004c,a). Enerzijds kan men het resultaat berekenen aan de hand van een momentopname van het AXML systeem, dus op basis van de documenten van het systeem in hun huidige staat. Hierbij wordt geen enkele service call gematerialiseerd. Anderzijds kan men zorgen dat alle relevante web services uitgevoerd zijn vóór de query geëvalueerd wordt. Dit noemt het volledige resultaat van een query en (enkel) hiervoor bespreken we hieronder een aantal strategieën.

- Het naïeve algoritme voert simpelweg alle service calls uit voor de query evaluatie. Dit leidt uiteraard tot een groot deel nutteloze data en een lange uitvoeringstijd. Tevens kan dit zorgen voor een oneindige berekening, terwijl het berekenen van het resultaat op een andere manier misschien wel mogelijk is. Daarenboven bestaat de kans dat bepaalde service calls uitgevoerd worden, die gewoon als functie in het resultaat van de query mogen voorkomen.

- Een minder naïeve strategie bestaat erin de query processor het document top-down te laten doorlopen en alle web services die gepasseerd worden uit te voeren. Hierdoor wordt slechts een klein deel van het document gematerialiseerd. Echter het gelijktijdig uitvoeren van query processing en service calls is zeer inefficiënt, gezien de query processor geblokkeerd wordt tijdens het uitvoeren van de service calls en men rekening moet houden met de uitvoer van de geactiveerde services.
- Bij lazy query evaluation worden enkel die service calls uitgevoerd die data leveren voor het resultaat van de query. Daarvoor wordt dus eerst het deel van het document bepaald waar de query zijn informatie haalt en enkel in dat deel worden de service calls uitgevoerd.
- Een andere strategie is de fire-once semantiek, met name elke service call wordt slechts één keer uitgevoerd, alvorens het resultaat van de query berekend wordt.

Bij het zoeken naar relevante calls moet men rekening houden met de performantie. Wanneer de kosten vrij hoog zijn voor het exact bepalen welke service calls noodzakelijk zijn, kan het beter uitkomen om met minder accurate berekeningen een groep services uit te voeren, waarin zeker de relevante calls zitten, maar mogelijk ook nog andere.

1.5.2 Uitvoering van de relevante calls

Gegeven dat alle relevante service calls voor een query gevonden zijn, moet nog een volgorde van uitvoeren bepaald worden. Na elke call moet opnieuw bekeken worden welke web service we aanroepen, gezien een relevante service niet noodzakelijk relevant blijft na de uitvoering van andere services. Bijvoorbeeld, als de query uit figuur 1.6 uitgevoerd wordt op het voorbeeld AXML document uit figuur 1.1, lijkt het logisch dat alle vermelde services relevant zijn voor de query. Echter wanneer de `geefRating` service in het linkse deel van het document minder dan vier sterren oplevert, is de `geefActeurs` service niet meer relevant.

1.6 Positive AXML en simple queries

Positive AXML, geïntroduceerd door Abiteboul *et al.* (2004c), is een speciale klasse van monotone AXML systemen waarbij de precieze functionaliteit van de web services gekend is. Dit in tegenstelling tot de black-box semantiek van algemene AXML systemen. Door deze restrictie kunnen bepaalde eigenschappen in verband met eindigheid bewezen worden, die in het algemene geval niet gelden. De web services zijn gedefinieerd door queries, die positive queries genoemd worden. Simple positive queries zijn queries die geen boomvariabelen gebruiken. Boomvariabelen worden gebruikt om een deelboom van een document voor te stellen. Een positive AXML systeem waarin alle functies gedefinieerd zijn door simple positive

queries is een simple positive AXML systeem. Deze systemen verbieden dus het kopiëren van deelbomen.

Positive queries zijn regels van de vorm *hoofd :- body*. De body voert een selectie uit analoog aan de *from* en *where* gedeeltes uit XQuery. Het hoofd komt overeen met het *return* gedeelte. Hieronder zien we een voorbeeld van een service in positive AXML. De service geeft de titels van vijfsterrenfilms geregisseerd door Quentin Tarantino.

```
titels{x} :- films{film{titel{x},
                        regisseur{"Quentin Tarantino"},
                        rating{"*****"}}}
```

Dit is een simple positive query, gezien de enige variabele x gebruikt wordt voor titels, dus strings van karakters. In sectie 1.5 hebben we een query gezien met deze syntax en zijn overeenkomstige boompatroonquery.

1.7 Semantiek

Over de semantiek van Active XML is tot hiertoe maar weinig geschreven. Abiteboul *et al.* (2004c) legt een aantal restricties op aan services en het uitvoeren ervan. Hierdoor kunnen ze garanderen dat de semantiek van documenten altijd uniek is, dus onafhankelijk van de volgorde van service activaties. Het geeft echter geen garantie dat de semantiek eindig is.

Het paper Active XML team (2003a) bespreekt de semantiek meer uitgebreid. Zij modelleren AXML peers als machines in een bepaalde toestand, die met elkaar interageren door middel van asynchrone communicatie. De toestand van het systeem is de vector van de toestanden van al zijn componenten. Een peer verandert van toestand telkens:
- een service call uit een document geactiveerd wordt,
- een document herschreven wordt omwille van het resultaat van een service of
- wanneer een andere peer één van zijn services aanroept.

Een AXML systeem evolueert daarom continu en ook niet-deterministisch. Het geheel van wijzigingen in een systeem is complex. Een aantal nuttige eigenschappen van computersystemen kunnen niet zomaar verondersteld worden in AXML, namelijk: service calls zijn niet altijd eindig, documenten kunnen oneindig groot worden, verschillende services kunnen elkaar beïnvloeden en services kunnen documenten misvormen zodat ze niet meer voldoen aan hun (DTD of XML) schema. Deze eigenschappen vormen niet noodzakelijk een probleem ingeval van web toepassingen. In andere omstandigheden kan het daarentegen nodig zijn om door restricties een aantal van de problemen op te lossen.

1.8 Historiek en verloop AXML project

Active XML documenten en services werden geïntroduceerd in Abiteboul *et al.* (2001). Een eerste versie van een AXML systeem werd gepresenteerd in Abiteboul *et al.* (2002). Het achterliggende idee, namelijk het mixen van functie calls en data, is niet nieuw. Het werd eerder al gebruikt in onder andere: stored procedures in relationele systemen, scripts in html of xml documenten, method calls in object georiënteerde databases,... Het nieuwe aan Active XML is dat zowel XML als web services gestandaardiseerd worden en dus AXML wereldwijd bruikbaar zal worden.

Vanaf eind 2003 worden er wereldwijd op regelmatige basis papers over Active XML tot congressen toegelaten. Sinds september 2004 is Active XML een open source project (Active XML team, 2004). Op de website van het Active XML project (Active XML team, 2003b) worden recente, nieuwe ontwikkelingen gemeld. Voor april 2006 meldt men dat het Distributed Query-Sub-Query Project draaiende is. Dit is een applicatie, gebaseerd op Active XML, met als doel het optimaliseren van Datalog queries in een peer-to-peer omgeving. Tevens wordt de ActiveXML Light-weight Client 1.0.1 gereleased. Deze tool levert een user interface voor Active XML en is een extensie van de Firefox webbrowser. Het vergemakkelijkt het creëeren en wijzigen van AXML documenten en laat toe vanuit de documenten web services aan te roepen.

1.9 Probleemstelling Thesis

Een eerste doelstelling van deze thesis is reeds vervat in dit eerste hoofdstuk, namelijk een grondige introductie leveren tot Active XML. Door het bestuderen van alle belangrijke aspecten van Active XML, beschikken we over voldoende achtergrondkennis om ons te verdiepen in de theoretische fundering ervan. Deze studie biedt de mogelijkheid om die fundering uit te breiden en te verbeteren. Een belangrijk resultaat is de bewijsvoering die staaft dat de semantiek van monotone Active XML systemen goed gedefinieerd is. Dit gegeven is uiteraard noodzakelijk, teneinde Active XML praktisch mogelijk te maken. In het laatste hoofdstuk behandelen we queries over Active XML systemen.

Nota: Mogelijke Modellen

Er zijn verschillende aspecten van Active XML die men kiest bij het vastleggen van een bepaald model. Zo wordt in Abiteboul *et al.* (2004c) een AXML document voorgesteld als een ongeordende gelabelde boom. Terwijl Abiteboul *et al.* (2004a); Milo *et al.* (2005) kiezen voor geordende bomen. Zoals eerder vermeld werd, zijn er ook verschillen inzake het opslaan van het resultaat van een web service. Abiteboul *et al.* (2004c) voegt het resultaat toe als sibling van de functieknoop en Abiteboul *et al.* (2004a) vervangt de functieknoop door het resultaat. Een ander aspect is het al dan niet kennen van de functionaliteit van web services. Standaard wordt een black-box semantiek gebruikt voor web services. Enkel bij Positive Active XML is dit niet het geval.
De rode draad door de verschillende modellen is de voorstelling van een AXML document als een gelabelde boom, met data- en functieknopen, waarbij de kinderen van functieknopen als parameters dienen voor de services.

Hoofdstuk 2

Formele definities en semantiek

Dit hoofdstuk is gebaseerd op sectie 2 van Abiteboul *et al.* (2004c). In die sectie worden AXML documenten en systemen formeel voorgesteld en nadien wordt de semantiek onder de loep genomen.

We beginnen dit hoofdstuk met het definiëren van het gebruikte model. Vervolgens geven we alle nodige formele definities om de semantiek van een Active XML systeem te kunnen beschrijven. Tenslotte geven we de aanzet om te bewijzen dat deze semantiek goed gedefinieerd is. De verdere uitwerking daarvan zal gebeuren in hoofdstukken 3 en 4, waarbij we hoofdstuk 4 afsluiten met het afgewerkte bewijs.

2.1 Model

In dit hoofdstuk gebruiken we het model uit Abiteboul *et al.* (2004c). We modelleren Active XML documenten als ongeordende, gelabelde bomen met twee soorten knopen, data- en functieknopen. Het resultaat van services wordt als sibling van de functieknoop toegevoegd aan het document. We gebruiken ook de black-box semantiek van services, met andere woorden de definitie van de services is ongekend.
Voor willekeurige web services kan de volgorde waarin service calls uitgevoerd worden een verschil uitmaken. Daarom worden volgende restricties opgelegd: web services moeten monotoon zijn, met andere woorden de documenten mogen enkel gewijzigd worden door data toe te voegen en de sequentie van functie aanroepen moet rechtvaardig zijn, wat betekent dat élke functie die nieuwe data teruggeeft, op een bepaald moment uitgevoerd zal worden.

2.2 Definities

We veronderstellen het bestaan van enkele disjuncte domeinen, namelijk een domein met knopen $\mathcal{N}$, met labels $\mathcal{L}$, met functienamen $\mathcal{F}$ en met atomaire waardes $\mathcal{V}$.

Op basis daarvan kunnen we de formele definitie geven van een AXML document, zoals in figuur 1.1.

Definitie 2.1. *Een ongeordend* **AXML document** *is een expressie* (T, λ)*, waarbij:*
- $T = (N,E)$ *een ongeordende boom is,*
- $N \subset \mathcal{N}$ *een eindige verzameling van knopen is,*
- $E \subset N \times N$ *de bogen zijn en*
- $\lambda : N \rightarrow \mathcal{L} \cup \mathcal{F} \cup \mathcal{V}$ *een functie is over knopen.*
Dit allemaal gegeven dat:
(i) alleen aan de bladeren atomaire waardes toegekend worden en
(ii) aan de wortel een label of een atomaire waarde toegekend wordt.

In het vervolg bedoelen we met een document altijd een ongeordend AXML document. $\lambda(n)$ is de markering van knoop n. Knopen met een markering in $\mathcal{L} \cup \mathcal{V}$ (dus labels of atomaire waardes) worden dataknopen genoemd en deze met een markering in $\mathcal{F}$ (de functienamen) noemen we functieknopen.

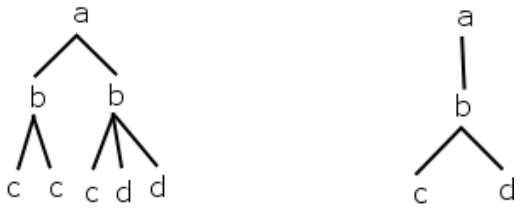
Definitie 2.2. *Een document* (T_1, λ_1) *is* **gesubsumeerd** *door een document* (T_2, λ_2)*, genoteerd als* $(T_1, \lambda_1) \subseteq (T_2, \lambda_2)$*, als er een afbeelding h bestaat die aan volgende eisen voldoet:*
- de knopen van T_1 *worden gemapt op de knopen van* T_2*,*
- de wortel van T_1 *wordt gemapt op de wortel van* T_2*,*
- de ouder-kind relaties tussen knopen blijven behouden en
- de markering van knopen blijft identiek, met andere woorden $\forall n : \lambda_1(n) = \lambda_2(h(n))$*.*

Als $(T_1, \lambda_1) \subseteq (T_2, \lambda_2)$ *waar is, spreken we over de* **subsumptie** *van* (T_1, λ_1) *in* (T_2, λ_2)*. Twee documenten* d_1 *en* d_2 *zijn* **equivalent***, genoteerd als* $d_1 \equiv d_2$*, als en slechts als zowel* $d_1 \subseteq d_2$ *als* $d_2 \subseteq d_1$ *geldt. Een document d noemt men* **gereduceerd** *als geen deelboom van d equivalent is aan d zelf. Een* **gereduceerde versie** *van een document d is een gereduceerd document d' equivalent aan d.*

Figuur 2.1 toont een voorbeeld van een niet gereduceerd document en de gereduceerde versie van datzelfde document. We vermelden zonder bewijs volgende eigenschap (in Abiteboul *et al.* (2004c) is een schets gegeven van het bewijs).

Eigenschap 2.3. *Document subsumptie is een transitieve en reflexieve relatie.*

De noties van subsumptie, equivalentie en reductie van documenten kunnen eenvoudig uitgebreid worden tot verzamelingen van documenten. Een bos φ is gesubsumeerd door een bos φ' als elke boom in φ gesubsumeerd is door een boom in φ'. Een bos φ is gereduceerd als daarin alle bomen gereduceerd zijn en geen enkele gesubsumeerd is door een andere.



Figuur 2.1: Een niet gereduceerd document en de gereduceerde versie ervan

In wat volgt identificeren we elk document of elke verzameling van documenten met zijn equivalentieklasse en gebruiken de gereduceerde versie als vertegenwoordiger van de klasse.
Document subsumptie negeert de semantiek van functies. Stel bijvoorbeeld dat voor twee functies f en g, voor elke mogelijke, geldige input x, $f(x) \subseteq g(x)$ geldt. Dan kunnen we de twee documenten `<a><sc service="f"> 5 </sc></a>` en `<a><sc service="g"> 5 </sc></a>` toch niet vergelijken.[1]

Definitie 2.4. *Een* **web service** *s over een verzameling documentnamen $\{d_1, \ldots, d_n\}$, gebruikmakend van een verzameling F met functienamen, wordt gedefinieerd als volgt. Gegeven een toekenning θ, die $d_1, \ldots, d_n$ afbeeldt op AXML documenten, gaat $s(\theta)$ een bos AXML documenten teruggeven met enkel functienamen in F.*

Informeel verstaan we hieruit dat een web service eender welke actie mag uitvoeren op de inputdocumenten, op voorwaarde dat de outputdocumenten enkel functienamen bevatten uit F. Web services mogen gebruik maken van twee gereserveerde documentnamen, *input* en *context*, die respectievelijk de parameters van de service en de context van de functieknoop voorstellen (dit wordt later exact gedefinieerd).

We focussen hier enkel op **monotone services**. Dit zijn services s zodat, voor elke θ, θ' en voor elke i, waarvoor $\theta(d_i) \subseteq \theta'(d_i)$ geldt, ook $s(\theta) \subseteq s(\theta')$ geldt. Dit betekent dat services de documenten enkel mogen wijzigen door data toe te voegen.

Definitie 2.5. *Een* **monotoon AXML systeem** *is een expressie (D, F, I) waarin*
- D een eindige set van documentnamen is (input en context zitten niet in D),
- F een eindige set van functienamen is,
- I een afbeelding over $D \cup F$ is, zodat voor elke $d \in D$, $I(d)$ een document is met alleen functienamen in F en voor elke $f \in F$, $I(f)$ een monotone service is over $D \cup \{input, context\}$ die de set functienamen F gebruikt.

[1] De syntax van deze documenten is analoog aan voorbeelden uit hoofdstuk 1, zie bijvoorbeeld figuur 1.1.

Heel precies veronderstellen we dat de documenten geen knopen gemeenschappelijk hebben. In wat volgt bedoelen we met een systeem altijd een monotoon AXML systeem. Als D en F duidelijk zijn uit de context, dan gebruiken we eenvoudigweg I om het systeem aan te duiden.

Beschouw een document d in een systeem I. Als een functieknoop v, gemarkeerd met label f, geactiveerd wordt, dan wordt $I(f)$ geëvalueerd. Om dit formeel te definiëren, geven we eerst een betekenis θ aan de documentnamen gebruikt door de functie.
De betekenis van **input**, $\theta(input)$, is de boom met als wortel een knoop gelabeld *input* en alle deelbomen onder v als kinderen. De betekenis van **context**, $\theta(context)$, is de deelboom uit d met als wortel de ouder van v. De betekenis van de documentnamen in D is gegeven door I (zie definitie 2.5).

Definitie 2.6. *Voor een systeem I zeggen we dat $I \xrightarrow{v} I'$ geldt, als I' bekomen wordt uit I door de invocatie van een functieknoop v in I. Deze transitie kan enkel gebeuren indien $I \not\equiv I'$ waar is. Een (mogelijk oneindige)* **herschrijvingssequentie** *is een sequentie $I \xrightarrow{v_1} I_1 \xrightarrow{v_2} I_2 \rightarrow \ldots \xrightarrow{v_n} I_n \ldots$ We zeggen dat I herschrijft tot I_n, genoteerd als $I \xrightarrow{*} I_n$. We zeggen dat het systeem eindigt in I_n als er geen functieknoop v_{n+1} in I_n is en geen I_{n+1} zodat $I_n \xrightarrow{v_{n+1}} I_{n+1}$ waar is. Een oneindige sequentie noemen we rechtvaardig als voor elke i en elke functieknoop $v \in I_i$ een $j > i$ bestaat zodat minstens één van volgende condities geldt:*
(i) $I_j \xrightarrow{v} I_{j+1}$ of (ii) een invocatie van v zou I_j niet wijzigen.

Merk op dat het goed mogelijk is dat een herschrijvingssequentie eindigt. Er komen dan op een bepaald moment gewoon geen nieuwe service calls meer op de proppen.
Gezien we enkel met monotone functies werken (zie paragraaf 2.1), volgt uit $I \xrightarrow{v} I'$ ook dat $I \subseteq I'$ waar is. In een rechtvaardige sequentie wordt dus élke functieknoop die data aan het document toevoegt, op een bepaald moment uitgevoerd.

Een **oneindig AXML document** is een document waarbij de verzameling knopen niet noodzakelijk eindig is. Een **oneindig monotoon AXML systeem** bekomen we door oneindige documenten toe te laten in een systeem. We blijven echter wel veronderstellen dat een eindig aantal documenten en functies gebruikt worden.

Definitie 2.7. *Stel I is een monotoon AXML systeem. De* **semantiek** *van I, genoteerd als $[\,I\,]$, is gedefinieerd als volgt:*
- ofwel geldt $[\,I\,] = J$ voor een eindig systeem J zodat $I \xrightarrow{*} J$ en het systeem eindigt in J,
- of $[\,I\,] = \cup\{I_i\}$ is waar voor een oneindige, rechtvaardige herschrijving $I \xrightarrow{v_1} I_1 \rightarrow \ldots \xrightarrow{v_i} I_i \ldots$, met andere woorden aan elke documentnaam in I wordt de kleinste bovengrens van de overeenkomstige documenten in de herschrijvingssequentie toegekend.

De semantiek van een document $d \in [\,I\,]$, genoteerd als $[\,d\,]$, is gegeven door $[\,I\,]$.

2.3 Eigenschappen semantiek

In deze sectie tonen we aan dat de semantiek goed gedefinieerd is en dus onafhankelijk is van de volgorde van functie aanroepen. De stelling en het gevolg uit deze sectie worden vermeld in Abiteboul *et al.* (2004c). Tevens wordt daarin een bewijs geschetst voor de stelling.

Stelling 2.8. *Stel I is een systeem en veronderstel $I \xrightarrow{*} J$ en $I \xrightarrow{*} K$, dan geldt:*
(i) als J eindigt in J', dan geldt $K \subseteq J'$ en
(ii) als $J \xrightarrow{v_1} J_1 \xrightarrow{v_2} J_2 \xrightarrow{v_3} \ldots$ een oneindige, rechtvaardige herschrijving is, dan geldt voor een bepaalde i, $K \subseteq J_i$.

Bewijs. Dit bewijs is per inductie op het aantal aangeroepen functies n in de herschrijvingssequentie die K genereert.

(i) Stel $n = 0$, dan moeten er geen functies uitgevoerd worden om K uit I te bekomen, met andere woorden $I \equiv K$. Uit $I \xrightarrow{*} J$ en definitie 2.6 (van herschrijvingssequentie) volgt dat $I \subseteq J$. We weten dat J eindigt in J'. Tevens weten we uit definitie 2.6 dat $J \subseteq J'$ waar is. Uit de transitiviteit van document subsumptie (zie eigenschap 2.3) volgt dan dat $I \subseteq J'$. De equivalentie van I en K leidt ons tot de conclusie dat in het basisgeval aan $K \subseteq J'$ voldaan is.
Als inductiehypothese beweren we dat de stelling geldt voor een systeem K', dat bekomen wordt uit I door het uitvoeren van $n-1$ functies. We weten nu dat voor een bepaalde functieknoop x, $K' \xrightarrow{x} K$ geldt. Omwille van de monotoniciteit van de functies moet in die stap nieuwe data toegevoegd wordt aan K'. Omdat $I \xrightarrow{*} J$ een rechtvaardige herschrijvingssequentie is, bestaat er een stap in deze herschrijving waarin functie x uitgevoerd wordt, namelijk er bestaat een natuurlijk getal i met $J_i \xrightarrow{x} J_{i+1}$. Hieruit verstaan we dat de nieuwe data door functie x geïntroduceerd, reeds in J' bevat zit. Uit deze laatste stap en de inductiehypothese kunnen we concluderen dat $K \subseteq J'$ waar is.

(ii) Stel opnieuw $n = 0$, dus $I \equiv K$. Hieruit volgt dat $K \subseteq J$ waar is en daardoor geldt $K \subseteq J_i$ voor eender welke i.
Als inductiehypothese nemen we opnieuw aan dat de stelling geldt voor een systeem K' bekomen uit I door uitvoering van $n-1$ functies. Voor een bepaalde functieknoop x geldt $K' \xrightarrow{x} K$. Gezien $J \xrightarrow{v_1} J_1 \xrightarrow{v_2} J_2 \xrightarrow{v_3} \ldots$ een rechtvaardige herschrijvingssequentie is, weten we dat er een j bestaat waarvoor geldt $J_j \xrightarrow{x} J_{j+1}$. Wanneer we nu i gelijkstellen aan $(j+1)$, dan kunnen we uit deze laatste stap en de inductiehypothese besluiten dat $K \subseteq J_i$. □

Gevolg 2.9. *De semantiek van monotone AXML systemen is goed gedefinieerd. Namelijk: (i) Als één herschrijving eindigt, dan eindigt elke herschrijving in hetzelfde eindige systeem. (ii) Als één herschrijving niet eindigt, dan eindigt geen enkele herschrijving en elke rechtvaardige herschrijving produceert hetzelfde oneindige systeem.*

In Abiteboul *et al.* (2004c) staat geschreven dat eenvoudig bewezen kan worden dat deze bewering volgt uit stelling 2.8. Het bleek echter niet evident om dit in detail uit te werken.

Bewijs. **(i)** Dit volgt duidelijk uit deel (i) van stelling 2.8. Gegeven dezelfde herschrijvingen $I \xrightarrow{*} J$ en $I \xrightarrow{*} K$, waarbij J eindigt in J' en dus de eerste herschrijving eindigt. Dan weten we uit stelling 2.8 dat voor de tweede herschrijving geldt dat $K \subseteq J'$. Omwille van dit laatste kan K geen andere functie calls bevatten als J' en is duidelijk dat K zal eindigen in hetzelfde systeem J'. Het omwisselen van de rollen van J en K leidt tot dezelfde conclusie.

(ii) Dit deel van het bewijs in detail uitwerken is verre van evident. We verdiepen ons in deze materie in hoofdstukken 3 en 4. Aan het einde van hoofdstuk 4 wordt dit bewijs afgewerkt. □

We zoeken de limiet van een rechtvaardige, oneindige herschrijving. Daarvoor schakelen we over van een AXML systeem naar één AXML document, wat we kunnen voorstellen als een boom. Gezien de redeneringen eenvoudiger zijn voor geordende bomen, zullen we daarmee beginnen, om vervolgens te veralgemenen naar ongeordende bomen. Daarna werken we weer verder met AXML documenten en systemen.

Hoofdstuk 3

Limiet van een rij geordende, eindige bomen

De redeneringen met geordende bomen zijn gebaseerd op de eerste twee paragrafen van het paper „Fundamental Properties of Infinite Trees" van B. Courcelle. (Courcelle, 1983) Merk op dat het model uit paragraaf 2.1 gebaseerd is op ongeordende bomen.

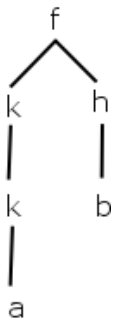
In eerste instantie wordt het concept van een geordende boom formeel gedefinieerd. Nadien wordt aangetoond dat de notie van subsumptie voor deze bomen een complete partiële orde is. Deze orde maakt het mogelijk aan te tonen dat een stijgende rij bomen een kleinste bovengrens heeft. Vervolgens concretiseren we deze bovengrens en tot slot tonen we aan dat een oneindige boom de limiet is van een rij eindige bomen.

3.1 Definities

Definitie 3.1. *Een* **geordende boom** *over een alfabet F is een partiële afbeelding $t : \mathbb{N}_0^* \to F$, zodat het domein een boomdomein is. Dit wil zeggen dat het voldoet aan volgende condities:*
- $Dom(t)$ is niet leeg en prefix-gesloten (met andere woorden als $\alpha, \beta \in \mathbb{N}_0^*$ en $\alpha\beta \in Dom(t)$, dan $\alpha \in Dom(t)$)
- als $\alpha \in \mathbb{N}_0^*$, $i, j \in \mathbb{N}_0$, $1 \leq i \leq j$ en $\alpha j \in Dom(t)$, dan $\alpha i \in Dom(t)$.
Wanneer F het speciale symbool Ω bevat, mag dit enkel voorkomen als label van een blad van de boom.

Stel $F = \{a, b, f, h, k\}$. Dan kunnen we een partiële afbeelding t definiëren als volgt: $t(\varepsilon) = f$, $t(1) = t(11) = k$, $t(111) = a$, $t(2) = h$, $t(21) = b$. Deze boom is voorgesteld in figuur 3.1. Hierin zien we dat het domein van t bestaat uit sequenties van natuurlijke getallen. $\mathbb{N}_0$ staat voor alle natuurlijke getallen zonder 0. $\mathbb{N}_0^*$ staat dan uiteraard voor 0 of meer dergelijke getallen, waarbij een sequentie van 0 getallen de lege string is, die we voorstellen met het symbool ε. Elke sequentie komt overeen met een bepaalde knoop in de boom. Bijvoorbeeld

$t(21)$ is het label van de knoop die gevonden wordt door vanaf de wortel, eerst de tak naar het tweede kind te volgen en vervolgens het eerste kind van die knoop te nemen. Dat is de knoop die we zoeken en is gelabeld met b.



Figuur 3.1: Voorbeeld van een geordende boom

Alfabet F is willekeurig te kiezen. In wat volgt veronderstellen we dat F reeds vastgelegd is. Merk op dat we met deze definitie een oneindige boom kunnen voorstellen. We definiëren $M^{\infty}(F)$ als de verzameling van alle willekeurige en dus mogelijk oneindige bomen over F. Gezien Ω een speciale rol speelt, gebruiken we ook de speciale notatie $M_{\Omega}^{\infty}(F)$ voor $M^{\infty}(F \cup \{\Omega\})$.

We voegen aan $M_{\Omega}^{\infty}(F)$ de notie van **subsumptie** toe.

Definitie 3.2. *Voor alle mogelijke bomen t en t' uit $M_{\Omega}^{\infty}(F)$ geldt:*

$$\mathbf{t \leq t'} \Leftrightarrow Dom(t) \subseteq Dom(t') \ \wedge \ (\forall \alpha \in Dom(t) : \ t(\alpha) \neq \Omega \ \Rightarrow \ t(\alpha) = t'(\alpha))$$

Intuïtief volgt hieruit dat we een boom t' kunnen transformeren in een boom t, met $t \leq t'$, door op een bepaalde plaats in boom t' een knoop af te hakken. Hiermee kunnen we het label van één knoop veranderen, van een bepaalde $f \in F$ naar Ω, of een hele deelboom wegsnoeien.

3.2 Subsumptie ($\leq$) is een complete partiële orde

Een partiële orde is een binaire relatie R over een bepaalde verzameling P, die reflexief, antisymmetrisch en transitief is.

Stelling 3.3. $\leq$ *(uit definitie 3.2) is een partiële orde.*

Bewijs. **reflexief** Voor elke willekeurige boom t in $M_{\Omega}^{\infty}(F)$, geldt $t \leq t$, want $Dom(t) = Dom(t)$ en voor elke α in het domein geldt dat $t(\alpha) = t(\alpha)$.

antisymmetrisch Neem twee bomen t, t' uit $M_\Omega^\infty(F)$ met $t \leq t'$ en $t' \leq t$. Dan geldt dat $Dom(t) \subseteq Dom(t')$ en $Dom(t') \subseteq Dom(t)$. Hieruit volgt dat $Dom(t) = Dom(t')$. Ook weten we dan dat $\forall \alpha \in Dom(t)$: $t(\alpha) \neq \Omega \Rightarrow t(\alpha) = t'(\alpha)$ en $\forall \alpha \in Dom(t')$: $t'(\alpha) \neq \Omega \Rightarrow t'(\alpha) = t(\alpha)$ geldt. Hieruit volgt onmiddellijk dat alle overeenkomende knopen met een label verschillend van Ω gelijk zijn. Gezien $Dom(t) = Dom(t')$ hebben beide bomen juist dezelfde vorm en een gelijk aantal knopen. Stel nu dat een knoop in t als label Ω heeft, dan heeft de overeenkomende knoop in t' hetzelfde label, want anders zou $\forall \alpha \in Dom(t')$: $t'(\alpha) \neq \Omega \Rightarrow t'(\alpha) = t(\alpha)$ niet waar zijn voor die knoop uit t'. Een analoge redenering waarbij de rollen van t en t' omgewisseld zijn, leidt ons tot het besluit dat t en t' gelijk zijn.

transitief Voor willekeurige bomen t, t' en t'' uit $M_\Omega^\infty(F)$ waarvoor geldt dat $t \leq t'$ en $t' \leq t''$, geldt $Dom(t) \subseteq Dom(t')$ en $Dom(t') \subseteq Dom(t'')$ en dus is ook $Dom(t) \subseteq Dom(t'')$ waar. Tevens weten we dat $\forall \alpha \in Dom(t)$: $t(\alpha) \neq \Omega \Rightarrow t(\alpha) = t'(\alpha)$ en $\forall \alpha \in Dom(t')$: $t'(\alpha) \neq \Omega \Rightarrow t'(\alpha) = t''(\alpha)$ waar zijn. Hieruit volgt dat $\forall \alpha \in Dom(t)$: $t(\alpha) \neq \Omega \Rightarrow t(\alpha) = t''(\alpha)$ geldt en dus is $t \leq t''$. □

We construeren nu een boom met één knoop gelabeld met Ω die we t_Ω noemen en we beweren dat dit de kleinst mogelijke boom is van $M_\Omega^\infty(F)$.

Eigenschap 3.4. *$\forall t \in M_\Omega^\infty(F)$: $t_\Omega \leq t$. Met andere woorden, (i) $Dom(t_\Omega) \subseteq Dom(t)$ en (ii) $\forall \alpha \in Dom(t_\Omega)$: $t_\Omega(\alpha) \neq \Omega \Rightarrow t_\Omega(\alpha) = t(\alpha)$*

Bewijs. (i) $Dom(t_\Omega) = \{\varepsilon\}$ en dus moeten we bewijzen dat ε een element is van $Dom(t)$. Uit de definitie van een boom volgt dat $Dom(t) \neq \emptyset$, met andere woorden $\exists \alpha \in Dom(t)$. ε is een prefix van elke string van karakters, dus ook van α. Gezien $Dom(t)$ prefix-gesloten is, geldt dat $\varepsilon \in Dom(t)$.
(ii) De enige α in $Dom(t_\Omega)$ is ε. Gezien de bewering voor de implicatie, $t_\Omega(\varepsilon) \neq \Omega$, niet waar is, valt er verder niets meer te bewijzen. □

We beschouwen nu een **stijgende rij** van bomen, met andere woorden een rij bomen $t_0, t_1, t_2, \ldots$, die we noteren als $(t_n)_{n \in \mathbb{N}}$, waarbij $t_i \leq t_{i+1}$, $\forall i \in \mathbb{N}$.

Definitie 3.5. *$\leq$ is* **compleet** *als elke stijgende rij een kleinste bovengrens heeft.*

Een stijgende rij heeft een kleinste bovengrens, als er een boom t bestaat zodat (i) $\forall i \in \mathbb{N}$: $t_i \leq t$ én (ii) voor elke andere bovengrens t' van deze rij, geldt $t \leq t'$. Uit (i) volgt dat t een bovengrens is en uit beide beweringen volgt dat het dé kleinste bovengrens is. Een kleinste bovengrens is altijd uniek, op equivalentie na.

We definiëren de kleinste bovengrens van een stijgende rij als de limiet van die rij.

Definitie 3.6. *Gegeven een stijgende rij* $(t_n)_{n\in\mathbb{N}}$*, definiëren we de* **limiet van een rij** *als* $\lim_{n\to\infty} t_n = t$*, waarbij* t *bepaald is door:*

$Dom(t) = \bigcup_{n\in\mathbb{N}} Dom(t_n)$ *en*

$$\forall \alpha \in Dom(t) : t(\alpha) = \begin{cases} f \in F & \text{als } t_i(\alpha) = f, \text{ voor een bepaalde } i \\ \Omega & \text{als } t_i(\alpha) = \Omega, \text{ voor elke } i \text{ waarvoor geldt } \alpha \in Dom(t_i) \end{cases}$$

Eigenschap 3.7. *De boom* t *die in definitie 3.6 geconstrueerd wordt als limiet van een stijgende rij, voldoet aan definitie 3.1 van een geordende boom.*

Bewijs. Hiervoor moeten we aantonen dat $Dom(t)$ een geldig boomdomein is. Dat het domein van t niet leeg kan zijn, is triviaal, gezien het domein van elke boom in de rij ook niet leeg kan zijn. De tweede voorwaarde is dat het domein prefix-gesloten is. Stel dat $\alpha, \beta \in \mathbb{N}_0^*$ en $\alpha\beta \in Dom(t)$. Dan moeten we aantonen dat α ook een element is van $Dom(t)$. $Dom(t) = \bigcup_{n\in\mathbb{N}} Dom(t_n)$, dus er bestaat een i waarvoor geldt $\alpha\beta \in Dom(t_i)$. Omdat t_i zelf een boomdomein is, zit α in $Dom(t_i)$. Dan zit α automatisch ook in $Dom(t)$. Dat t ook aan de laatste conditie voor een boomdomein voldoet, is analoog na te gaan. □

Stelling 3.8. *De limiet van een stijgende rij* $\lim_{n\to\infty} t_n = t$*, uit definitie 3.6, is de kleinste bovengrens van die rij.*

Bewijs. Zoals direct na definitie 3.5 vermeld wordt, moet er voor een kleinste bovengrens aan twee voorwaarden voldaan worden.

(i) Vooreerst moet voor boom t aan volgende voorwaarde voldaan zijn, $\forall i \in \mathbb{N}$: $t_i \leq t$. Uit definitie 3.2 weten we dat daarvoor volgende twee voorwaarden nagezien moeten worden voor elke i: $Dom(t_i) \subseteq Dom(t)$ en $\forall \alpha \in Dom(t_i) : t_i(\alpha) \neq \Omega \Rightarrow t_i(\alpha) = t(\alpha)$. Gezien $Dom(t)$ de oneindige unie neemt van de domeinen van alle bomen t_i uit de sequentie, is aan de eerste conditie voldaan.
Stel dat voor een bepaald natuurlijk getal i en een bepaalde string α uit het domein, $t_i(\alpha) \neq \Omega$, bijvoorbeeld $t_i(\alpha) = g \in F$. Dan is per definitie van t, $t(\alpha) = g$. Dus is ook aan de tweede voorwaarde voldaan.

(ii) Opdat t de kleinste bovengrens zou zijn, moet verder voor elke andere bovengrens t' van deze rij gelden dat $t \leq t'$. Neem een bovengrens t'. We moeten aantonen dat $t \leq t'$. Opnieuw moeten twee voorwaarden gecheckt worden: $Dom(t) \subseteq Dom(t')$ en $\forall \alpha \in Dom(t) : t(\alpha) \neq \Omega \Rightarrow t(\alpha) = t'(\alpha)$. Neem een willekeurige α in $Dom(t)$.

Gezien $Dom(t) = \bigcup_{n\in\mathbb{N}} Dom(t_n)$, bestaat er een natuurlijk getal i, waarvoor geldt $\alpha \in Dom(t_i)$. Omdat t' een bovengrens is, geldt $t_i \leq t'$. Hierdoor weten we dat $Dom(t_i) \subseteq Dom(t')$. Dus α zit in $Dom(t')$ en $Dom(t) \subseteq Dom(t')$.
Neem een α in $Dom(t)$ met $t(\alpha) \neq \Omega$. Stel $t(\alpha) = g \in F$. Hiervoor moeten we aantonen dat $t(\alpha) = g = t'(\alpha)$. Per definitie van t, weten we dat er een natuurlijk getal i bestaat waarvoor geldt $t_i(\alpha) = g$. Omdat t' een bovengrens is, geldt $t_i \leq t'$ en daardoor weten we dat $t'(\alpha) = g$. □

Uit deze stelling en definitie 3.5 van compleetheid kunnen we nu het volgende besluiten:

Gevolg 3.9. $\leq$ *is een complete partiële orde.*

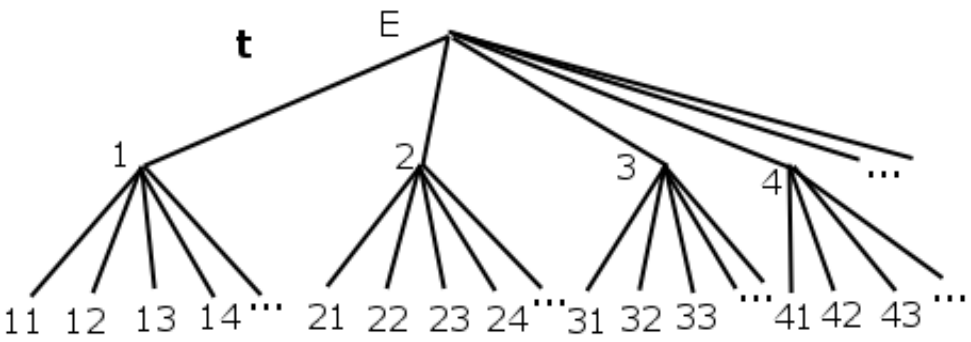
3.3 Oneindige boom, limiet van rij eindige, geordende bomen

We noteren de lengte van string α met $|\alpha|$.

Definitie 3.10. *Gegeven een willekeurige boom $t \in M_\Omega^\infty(F)$ en een natuurlijk getal n.*
Dan definiëren we de **boom t beperkt tot n**, *genoteerd als $t|_n$, als volgt:*
$Dom(\, t|_n\,) = \{\varepsilon\} \cup \{\, \alpha \in Dom(t) \mid \alpha$ *bestaat uit maximum n getallen uit het interval* $[1, n]\,\}$

$$en \quad \forall \alpha \in Dom(t|_n) : t|_n(\alpha) = \begin{cases} t(\alpha) & als \;\; |\alpha| < n \\ \Omega & als \;\; |\alpha| = n \end{cases}$$

In figuur 3.2 willen we een voorbeeld geven van een boom die zowel oneindig is in de breedte als in de lengte. Bij elke knoop staat het overeenkomstige element uit het domein in plaats van het label van de knoop. In plaats van symbool ε, voor de lege string, is een hoofdletter E gebruikt.



Figuur 3.2: Voorbeeld van een oneindige boom t

Voor de boom t kunnen we volgende domeinen bepalen, $Dom(t|_0) = \{\varepsilon\},\ Dom(t|_1) = \{\varepsilon, 1\}$, $Dom(t|_2) = \{\varepsilon, 1, 11, 12, 2, 21, 22\}$. Stel we nemen een bepaalde $i \geq 0$, dan bevat $t|_i$ enkel knopen uit de niveaus $0, 1, \ldots, i$. Verder bevat die voor alle knopen waarvan we de kinderen beschouwen (knopen op niveaus $0, 1, \ldots, (i-1)$) enkel de eerste i kinderen.

$t|_n$ bevat een deel van de knopen van t. Deze knopen worden identiek overgenomen van t, behalve de knopen op niveau n, want deze krijgen in $t|_n$ als label Ω. Stel nu dat we een eindige boom t hebben, die uit vijf niveaus bestaat en waarbij de knoop met de meeste kinderen, vijf kinderen heeft. Dan is t gelijk aan $t|_6$ of $t|_7$ of ... De boom $t|_5$ bevat ook alle knopen van t maar, ongeacht of niveau 5 het laagste niveau is of niet, krijgen alle knopen daar als label Ω. Dit toont dat alle knopen onder niveau 5 afgekapt worden en niet in de deelboom zitten. Uiteraard is het wel de bedoeling om de definitie van $t|_n$ enkel te gebruiken voor oneindige bomen.

Eigenschap 3.11. *De geconstrueerde boom $t|_n$ (uit definitie 3.10) voldoet aan definitie 3.1 van een geordende boom.*

Bewijs. We moeten aantonen dat $Dom(t|_n)$ een geldig boomdomein is. $Dom(t|_n)$ mag dus niet leeg zijn, maar dat kan ook niet want het bevat in elk geval ε. Vervolgens moeten we checken of $Dom(t|_n)$ prefix-gesloten is. Neem $\alpha, \beta \in \mathbb{N}_0^*$ met $\alpha\beta$ in $Dom(t|_n)$, met andere woorden de lengte van $\alpha\beta$ is maximum n en de string bestaat enkel uit cijfers van 1 tot en met n. Als $\alpha\beta$ in $Dom(t|_n)$ zit, dan zit het ook in $Dom(t)$. $Dom(t)$ is zelf een boomdomein, dus het bevat ook α. Gezien α een substring is van $\alpha\beta$ voldoet het automatisch ook aan de voorwaarden voor $Dom(t|_n)$ en geldt $\alpha \in Dom(t|_n)$. De laatste voorwaarde gaat als volgt: $\alpha \in \mathbb{N}_0^*,\ \mathrm{i}, \mathrm{j} \in \mathbb{N},\ 1 \leq i \leq j$ en $\alpha j \in Dom(t|_n)$, dan $\alpha i \in Dom(t|_n)$. Neem een bepaalde string αj in $Dom(t|_n)$. De lengte van αj is maximum n en de string bestaat enkel uit cijfers van 1 tot en met n. αj moet in $Dom(t)$ zitten en gezien dat een boomdomein is bevat het ook αi. De lengte van αj en αi is gelijk. Aangezien $i \leq j$ bestaat αi ook enkel uit cijfers van 1 tot en met n, dus kunnen we besluiten dat αi in $Dom(t|_n)$ zit. □

Eigenschap 3.12. *$t|_0$, $t|_1$, $t|_2$, ... is een stijgende rij. Met andere woorden $\forall n \in \mathbb{N}$:*
$t|_n \leq t|_{n+1}$.

Bewijs. Neem een willekeurige $n \geq 0$.
- $Dom(t|_n) = \{\varepsilon\} \cup \{\ \alpha \in Dom(t) \mid \alpha$ bestaat uit maximum n getallen uit $[1, n]\ \}$
- $Dom(t|_{n+1}) = \{\varepsilon\} \cup \{\ \alpha \in Dom(t) \mid \alpha$ bestaat uit maximum $n+1$ getallen uit $[1, n+1]\ \}$

Hieruit volgt onmiddellijk dat $Dom(t|_n) \subseteq Dom(t|_{n+1})$.
Voor alle α in $Dom(t|_n)$ moet gelden dat als $t|_n(\alpha) \neq \Omega$ dan $t|_n(\alpha) = t|_{n+1}(\alpha)$. Uit de

definitie van $t|_n$ weten we dat voor alle α in $Dom(t|_n)$ met $|\alpha| < n,\ t|_n(\alpha) = t(\alpha)$. Voor $t|_{n+1}$ geldt dit zelfs voor strings α van lengte $n+1$, dus voor strings α met lengte kleiner dan n geldt $t|_n(\alpha) = t|_{n+1}(\alpha)$. De overblijvende strings uit $Dom(t|_n)$ hebben lengte n en daarvoor is $t|_n(\alpha) = \Omega$, dus moet niets meer bewezen worden. □

Gegeven een oneindige boom t, zoals deze in figuur 3.2, bevatten de bomen $t|_0$, $t|_1$, $t|_2, \ldots$ een steeds groter deel knopen van t. Dat de limiet van deze rij t zelf is, is intuïtief duidelijk. We tonen dit nu formeel aan en gebruiken daarvoor volgend lemma.

Lemma 3.13. *Voor een willekeurige boom t uit $M_\Omega^\infty(F)$ en een willekeurig getal $n \in \mathbb{N}$ geldt dat $t|_n$ een eindige boom is.*

Bewijs. Stel we nemen een bepaalde boom t en een getal n. Het volstaat aan te tonen dat het domein van $t|_n$ eindig is. Elke string uit het domein heeft lengte maximum n. Een string α bestaat uit getallen uit het interval $[1, n]$. Het domein kan dus uit maximum n^n verschillende strings bestaan en is daarom eindig. □

Stelling 3.14. *Elke oneindige boom is de limiet van een rij eindige bomen.*

Bewijs. Neem een willekeurige oneindige boom t. Voor t kunnen we de stijgende rij $t|_0$, $t|_1$, $t|_2, \ldots$ construeren. Uit lemma 3.13 weten we dat elke boom $t|_i$ uit die rij eindig is. Elke stijgende rij bomen heeft een kleinste bovengrens, namelijk de limiet (zie paragraaf 3.2). Definitie 3.6 geeft vorm aan deze limiet.
Voor $(t|_n)_{n\in\mathbb{N}}$ is de limiet $\lim_{n\to\infty} t|_n = t'$, waarbij t' bepaald is door:
$Dom(t') = \bigcup_{n\in\mathbb{N}} Dom(t|_n)$

$$en \quad \forall \alpha \in Dom(t') : t'(\alpha) = \begin{cases} f \in F & \text{als } t|_i(\alpha) = f, \text{ voor een bepaalde i} \\ \Omega & \text{als } t|_i(\alpha) = \Omega, \text{ voor elke i met } \alpha \in Dom(t|_i) \end{cases}$$

Wanneer we nu aantonen dat t en t' gelijk zijn, is de stelling bewezen. Eerst tonen we aan dat $Dom(t) = Dom(t')$.
Neem een willekeurige $\alpha \in Dom(t)$. Stel de lengte van α gelijk aan n en stel het grootste getal uit de string α gelijk aan m. Wanneer we nu definitie 3.10 (van $t|_n$) toepassen voor onze boom t en n gelijk aan $max(n, m)$, dan bekomen we $Dom(t|_{max(n,m)}) = \{\varepsilon\} \cup \{\ \alpha \in Dom(t)\ |\ \alpha$ bestaat uit maximum max(n,m) getallen uit het interval $[1, max(n, m)]\ \}$. Dit domein bevat zeker onze string α. $Dom(t') = \bigcup_{n\in\mathbb{N}} Dom(t|_n)$, dus α zit automatisch ook in $Dom(t')$ en dus geldt $Dom(t) \subseteq Dom(t')$.
Uit definitie 3.10 (van $t|_n$) volgt duidelijk dat voor elke mogelijke $n \in \mathbb{N}$, $Dom(t|_n) \subseteq Dom(t)$. Gezien $Dom(t') = \bigcup_{n\in\mathbb{N}} Dom(t|_n)$ geldt dan ook $Dom(t') \subseteq Dom(t)$. En dus is $Dom(t) = Dom(t')$ bewezen.

Vervolgens moeten we aantonen dat voor elke string α uit het gemeenschappelijke domein $t(\alpha) = t'(\alpha)$.
Neem een willekeurige α uit het domein en stel opnieuw n gelijk aan de lengte en m aan het grootste getal in α. In de boom $t|_{max(n,m)+1}$ kan de knoop door α bepaald zich niet op het laagste niveau bevinden, want $|\alpha| = n$ en het laagste niveau is $max(n, m) + 1$. Dus geldt $t|_{max(n,m)+1}(\alpha) = t(\alpha)$ ook als $t|_{max(n,m)+1}(\alpha)$ gelijk is aan Ω.
Stel nu dat $t|_{max(n,m)+1}(\alpha)$ gelijk is aan een $f \in F$. Dan is $t'(\alpha) = f$, want dan bestaat er een i namelijk $max(n, m) + 1$ waarvoor geldt $t|_i(\alpha) = f$.
Als $t|_{max(n,m)+1}(\alpha)$ gelijk is aan Ω, dan moeten we aantonen dat $\forall i \in \mathbb{N}_0$ met $\alpha \in Dom(t|_i)$ geldt dat $t|_i(\alpha) = \Omega$, opdat ook $t'(\alpha)$ gelijk zou zijn aan Ω. Nu weten we uit de definitie van $t|_n$ (definitie 3.10), uit het gegeven dat $t|_0$, $t|_1$, $t|_2, \ldots$ een stijgende rij is en uit de definitie van subsumptie (definitie 3.2), dat het label van een knoop uit een boom uit de rij, neem $t|_j(\beta)$, enkel kan veranderen van Ω in een label $f \in F$ en niet omgekeerd. Verder kan die labelverandering enkel optreden indien β een knoop voorstelt op het laagste niveau van $t|_j$. Aangezien we weten dat de knoop in $t|_{max(n,m)+1}$ door α bepaald, niet op het laagste niveau zit, geldt inderdaad voor alle bomen $t|_i$ uit de stijgende rij dat $t|_i(\alpha)$ gelijk is aan Ω.
Nu hebben we aangetoond dat voor alle $\alpha \in Dom(t) = Dom(t')$, geldt dat $t(\alpha) = t|_{max(n,m)+1}(\alpha) = t'(\alpha)$. Hieruit kunnen we besluiten dat boom t gelijk is aan boom t'. □

Hoofdstuk 4

Limiet van een rij ongeordende, eindige bomen

In dit hoofdstuk beschouwen we ongeordende bomen. Dit leunt al dichter aan bij de AXML documenten en systemen uit Abiteboul *et al.* (2004c), gezien hun model AXML documenten voorstelt als ongeordende bomen.

De structuur van dit hoofdstuk vertoont vele gelijkenissen met die van het vorige. Ditmaal bestuderen we ongeordende bomen. Deze modelwijziging verhoogt de complexiteit en vergt heel wat aanpassingen aan onze strategie. Tot slot zullen we het bewijs dat de semantiek van monotone AXML systemen goed gedefinieerd is, op punt stellen.

4.1 Definities

De definitie voor ongeordende bomen is exact gelijk aan definitie 3.1 voor geordende bomen.

Definitie 4.1. *Een* **ongeordende boom** *over alfabet F is een partiële afbeelding $t : \mathbb{N}_0^* \to F$, zodat het domein een boomdomein is. Dit wil zeggen dat het voldoet aan volgende condities:*
- $Dom(t)$ is niet leeg en prefix-gesloten (met andere woorden als $\alpha, \beta \in \mathbb{N}_0^*$ en $\alpha\beta \in Dom(t)$, dan $\alpha \in Dom(t)$)
- als $\alpha \in \mathbb{N}_0^*$, $i, j \in \mathbb{N}$, $1 \leq i \leq j$ en $\alpha j \in Dom(t)$, dan $\alpha i \in Dom(t)$.
Wanneer F het speciale symbool Ω bevat, mag dit enkel voorkomen als label van een blad van de boom.

Uiteraard is er bij ongeordende bomen meer vrijheid qua positionering van knopen. Bomen die we hierdoor als "gelijk" beschouwen, noemen we isomorfe bomen en zullen we verderop exact definiëren.

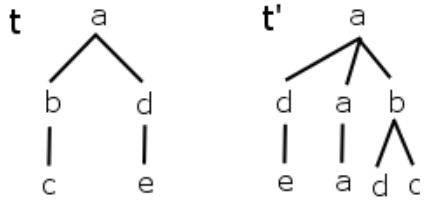
Voor ongeordende bomen is een nieuwe definitie van subsumptie noodzakelijk. Merk op dat deze heel erg verschilt met definitie 3.2 bij geordende bomen.

Definitie 4.2. *Voor alle mogelijke bomen t en t' uit $M_\Omega^\infty(F)$ geldt:*
$\mathbf{t \leq t'}$ *als er een homomorfisme bestaat van t naar t'.*
h is een homomorfisme van t naar t' als:

1. $h : Dom(t) \to Dom(t')$,
2. $h(\varepsilon) = \varepsilon$,
3. $\forall \alpha \in Dom(t)$: $t(\alpha) \neq \Omega \Rightarrow t(\alpha) = t'(h(\alpha))$ *en*
4. $\forall \alpha \in Dom(t)$, $\forall i \in \mathbb{N}_0$ *met* $\alpha i \in Dom(t)$, $\exists\, j \in \mathbb{N}_0$: $h(\alpha i) = h(\alpha) j$

Conditie 1 is voor de hand liggend. De tweede voorwaarde eist dat de wortel van t afgebeeld wordt op de wortel van t' en zegt daarbij niets over de labels van de wortels. De derde eis stelt dat een knoop α in t afgebeeld wordt op een knoop $h(\alpha)$ in t', waarbij dus beide knopen hetzelfde label dragen. Analoog aan definitie 3.2 van subsumptie bij geordende bomen, worden hierbij knopen gelabeld met Ω buiten beschouwing gelaten. Verder moet een homomorfisme de structuur van de ouder-kind relaties behouden. Met andere woorden voor alle knopen in t die ook in t' zitten, moet het pad vanuit de wortel identiek zijn. Dit ligt vast in de laatste conditie, waarin wordt geëist dat alle kinderen van knoop α in t afgebeeld worden op kinderen van $h(\alpha)$ in t'.

We zien in figuur 4.1 twee bomen t en t', waarvoor $t \leq t'$ geldt. Merk op dat t' meer knopen bevat als t.



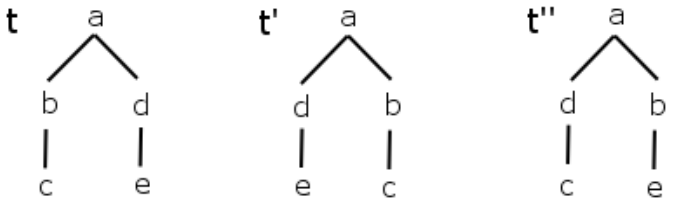
Figuur 4.1: $t \leq t'$, er bestaat dus een homomorfisme van t naar t'

Definitie 4.3. *Twee bomen t en t' zijn* **isomorf***, genoteerd als $t \cong t'$, als en slechts als er een bijectieve afbeelding ι bestaat van $Dom(t)$ naar $Dom(t')$ waarbij zowel ι als het inverse ι^{-1} homomorfismes zijn.*[1] *De bijectie ι noemen we het isomorfisme tussen t en t'.*

Figuur 4.2 illustreert deze definitie. Bomen t en t' in de figuur zijn isomorf. Het verschil tussen de twee bomen is dat in t' de kinderen van de wortel omgewisseld zijn. t' en t'' zijn niet isomorf. In t' is de knoop met label e een kind van de knoop met label d en in t'' is diezelfde knoop een kind van de knoop met label b. Gezien een homomorfisme een afbeelding is die de structuur behoudt, kunnen er geen homomorfismes bestaan tussen t' en t'' en bijgevolg

[1] De Griekse letter ι wordt uitgesproken als iota.

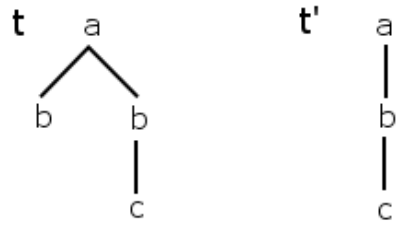
ook geen isomorfisme. Dit geven we aan door $t' \ncong t''$. Dus bij ongeordende bomen mogen knopen op een bepaald niveau van plaats verwisseld worden, zo lang daarbij de ouder-kind relaties intact blijven.



Figuur 4.2: t en t' zijn isomorf t' en t'' zijn niet isomorf

4.2 Complete partiële orde voor ongeordende bomen

Voor deze definitie van subsumptie is $\leq$ geen partiële orde. Figuur 4.3 levert een tegenvoorbeeld voor anti-symmetrie ingeval van ongeordende bomen. $t \leq t'$ en $t' \leq t$ maar $t \ncong t'$. Van t naar t' en omgekeerd bestaan homomorfismes, maar we kunnen geen bijectie definiëren tussen t en t'. Inderdaad zo zien we bijvoorbeeld dat beide knopen met label b uit t gemapt moeten worden op één knoop met label b in t', dus het homomorfisme van t naar t' kan niet injectief zijn.



Figuur 4.3: Tegenvoorbeeld anti-symmetrie

Definitie 4.4. *Twee bomen t_1 en t_2 zijn* **equivalent**, *genoteerd als $t_1 \equiv t_2$, als en slechts als voor deze bomen $t_1 \leq t_2$ en $t_2 \leq t_1$ geldt.*

t en t' uit figuur 4.3 zijn dus niet isomorf maar wel equivalent. Zoals we meteen zullen aantonen, bekomen we wel een partiële orde wanneer we overschakelen van aparte bomen naar equivalentieklassen van bomen.

Definitie 4.5. *De* **equivalentieklasse** *van een boom t_1, genoteerd als $\overline{t_1}$, is de verzameling van alle bomen equivalent aan t_1.*

We definiëren subsumptie nu op deze equivalentieklassen.

Definitie 4.6. $\overline{\mathbf{t_1}} \leq \overline{\mathbf{t_2}}$ *als* $t_1 \leq t_2$.

Eigenschap 4.7. *De definitie van* $\overline{t_1} \leq \overline{t_2}$ *hangt niet af van de gekozen representatieve bomen uit de equivalentieklassen.*

Bewijs. Neem een willekeurige boom t_3 uit equivalentieklasse $\overline{t_1}$ en een willekeurige boom t_4 uit $\overline{t_2}$. Voor deze twee bomen moeten we aantonen dat $t_3 \leq t_4$ geldt. We weten dat $t_3 \equiv t_1$. Daaruit volgt dat $t_3 \leq t_1$. Analoog weten we dat $t_2 \equiv t_4$ en dus $t_2 \leq t_4$. Met andere woorden, we bekomen volgende uitdrukking $t_3 \leq t_1 \leq t_2 \leq t_4$ en $t_3 \leq t_4$ is bewezen. □

Stelling 4.8. $\leq$ *gedefinieerd op de geïntroduceerde equivalentieklassen is een partiële orde.*

Bewijs. **reflexief** Voor elke equivalentieklasse $\overline{t_1}$ geldt $\overline{t_1} \leq \overline{t_1}$, want voor alle bomen t_2, t_3 uit $\overline{t_1}$ geldt $t_2 \equiv t_3$ en dus ook $t_2 \leq t_3$.

antisymmetrisch Gegeven twee equivalentieklassen $\overline{t_1}$ en $\overline{t_2}$, waarvoor $\overline{t_1} \leq \overline{t_2}$ en $\overline{t_2} \leq \overline{t_1}$ geldt, moeten we aantonen dat $\overline{t_1} = \overline{t_2}$. $\overline{t_1} \leq \overline{t_2}$ geldt als $t_1 \leq t_2$ en uit $\overline{t_2} \leq \overline{t_1}$ volgt $t_2 \leq t_1$. Hieruit volgt dat t_1 en t_2 equivalent zijn. Gezien $\overline{t_1}$ alle bomen bevat equivalent met t_1, zit ook t_2 daarin. De equivalentierelatie is duidelijk transitief, dus $\overline{t_1}$ bevat tevens alle bomen equivalent met t_2. Uit een analoge redenering vinden we dat $\overline{t_2}$ alle bomen equivalent met t_1 bevat. Dus kunnen we besluiten dat de twee equivalentieklassen gelijk zijn.

transitief We moeten aantonen dat uit $\overline{t_1} \leq \overline{t_2}$ en $\overline{t_2} \leq \overline{t_3}$ volgt dat $\overline{t_1} \leq \overline{t_3}$. $\overline{t_1} \leq \overline{t_2}$ geldt als $t_1 \leq t_2$ en dit geldt als er een homomorfisme h_1 bestaat van t_1 naar t_2. Analoog weten we uit $\overline{t_2} \leq \overline{t_3}$ dat $t_2 \leq t_3$ en een homomorfisme h_2 bestaat van t_2 naar t_3. We construeren nu de samenstelling $h_2 \circ h_1$ en tonen aan dat deze een homomorfisme is van t_1 naar t_3.

- $h_2 \circ h_1 : Dom(t_1) \to Dom(t_3)$?
  Gegeven dat $h_1 : Dom(t_1) \to Dom(t_2)$ en $h_2 : Dom(t_2) \to Dom(t_3)$ klopt dit.
- $(h_2 \circ h_1)(\varepsilon) = \varepsilon$?
  Gegeven dat $h_1(\varepsilon) = \varepsilon$ en $h_2(\varepsilon) = \varepsilon$ kunnen we het volgende afleiden $(h_2 \circ h_1)(\varepsilon) = h_2(h_1(\varepsilon)) = h_2(\varepsilon) = \varepsilon$.
- $\forall \alpha \in Dom(t_1)$: $t_1(\alpha) \neq \Omega \Rightarrow t_1(\alpha) = t_3(h_2(h_1(\alpha)))$?
  We weten dat: (i) $\forall \alpha \in Dom(t_1)$: $t_1(\alpha) \neq \Omega \Rightarrow t_1(\alpha) = t_2(h_1(\alpha))$
  en (ii) $\forall \alpha \in Dom(t_2)$: $t_2(\alpha) \neq \Omega \Rightarrow t_2(\alpha) = t_3(h_2(\alpha))$.

Neem een α uit $Dom(t_1)$ met $t_1(\alpha) \neq \Omega$. Uit (i) weten we dat $t_1(\alpha) = t_2(h_1(\alpha))$. Nu is $h_1(\alpha)$ een element van $Dom(t_2)$, dus weten we uit (ii) dat $t_2(h_1(\alpha)) = t_3(h_2(h_1(\alpha)))$.

- $\forall \alpha \in Dom(t_1),\ \forall i \in \mathbb{N}_0$ met $\alpha i \in Dom(t_1),\ \exists\, j \in \mathbb{N}_0$: $h_2(h_1(\alpha i)) = h_2(h_1(\alpha))j$?
  We weten dat:
  (i) $\forall \alpha \in Dom(t_1),\ \forall i \in \mathbb{N}_0$ met $\alpha i \in Dom(t_1),\ \exists\, j' \in \mathbb{N}_0$: $h_1(\alpha i) = h_1(\alpha)j'$
  en (ii) $\forall \alpha \in Dom(t_2),\ \forall i \in \mathbb{N}_0$ met $\alpha i \in Dom(t_2),\ \exists\, j'' \in \mathbb{N}_0$: $h_2(\alpha i) = h_2(\alpha)j''$.
  Neem een α uit $Dom(t_1)$ en een getal i zodat $\alpha i \in Dom(t_1)$. Uit (i) kunnen we volgende stap afleiden $h_2(h_1(\alpha i)) = h_2(h_1(\alpha)j')$, voor een bepaalde j' uit $\mathbb{N}_0$. $h_1(\alpha)$ zit in $Dom(t_2)$. Als we nu in (ii) voor α de string $h_1(\alpha)$ nemen en voor i het getal j', dan bekomen we $h_2(h_1(\alpha)j') = h_2(h_1(\alpha))j''$, voor een bepaalde j'' uit $\mathbb{N}_0$. □

Opnieuw construeren we de boom t_Ω, met één knoop gelabeld Ω en tonen aan dat dit de kleinst mogelijke boom is.

Eigenschap 4.9. $\forall t \in M_\Omega^\infty(F)$: $\overline{t_\Omega} \leq \overline{t}$.

Bewijs. Uit definitie 4.6 (subsumptie op equivalentieklassen) volgt dat we voor deze stelling moeten aantonen dat $t_\Omega \leq t$ waar is voor elke mogelijke boom t. Neem een willekeurige boom t. Hiervoor moeten we volgens definitie 4.2 (subsumptie op ongeordende bomen) aantonen dat een homomorfisme bestaat van t_Ω naar t. Gezien $Dom(t_\Omega) = \{\varepsilon\}$ volstaat het enkel voor ε het beeld te bepalen. We stellen $h(\varepsilon)$ gelijk aan ε.
Nu rest ons enkel te bewijzen dat afbeelding h een homomorfisme is. Het domein van t_Ω bevat enkel ε en dat element wordt op zichzelf afgebeeld. Zoals eerder aangetoond moet elk boomdomein minstens ε bevatten, dus h is een afbeelding van $Dom(t_\Omega)$ naar $Dom(t)$. Uiteraard is $h(\varepsilon) = \varepsilon$ ook waar. Als derde voorwaarde moet het volgende gelden: $\forall \alpha \in Dom(t_\Omega)$: $t_\Omega(\alpha) \neq \Omega \Rightarrow t_\Omega(\alpha) = t(h(\alpha))$. Voor de boom t_Ω bestaat geen element uit het domein waarvoor de conditie voor de implicatie waar is, dus voldoet h hier automatisch aan. Als laatste moeten we checken dat $\forall \alpha \in Dom(t_\Omega),\ \forall i \in \mathbb{N}_0$ met $\alpha i \in Dom(t_\Omega),\ \exists\, j \in \mathbb{N}_0$: $h(\alpha i) = h(\alpha)j$. De enige string in $Dom(t_\Omega)$ heeft lengte 0, dus er bestaat geen αi met $i \in \mathbb{N}_0$ die in $Dom(t_\Omega)$ zit. □

Wanneer we bewijzen dat elke stijgende rij $(\overline{t_n})_{n\in\mathbb{N}}$ een kleinste bovengrens heeft, dan is aangetoond dat $\leq$ voor equivalentieklassen een complete partiële orde is (zie definitie 3.5 van compleetheid). Gezien we dit gegeven enkel gebruiken voor stijgende rijen van eindige bomen, beperken we onze redenering in die zin. Dus $(\overline{t_n})_{n\in\mathbb{N}}$ is de oneindige rij $\overline{t_0}, \overline{t_1}, \overline{t_2}, \ldots$, waarbij $\forall i \in \mathbb{N}$: $\overline{t_i} \leq \overline{t_{i+1}}$ en t_i is eindig. Elk element van de rij is een verzameling van equivalente bomen. We definiëren de kleinste bovengrens van een stijgende rij als de limiet van die rij.

Definitie 4.10. *Stel*

$$m_n = \sum_{j<n} \#_c(t_j)$$

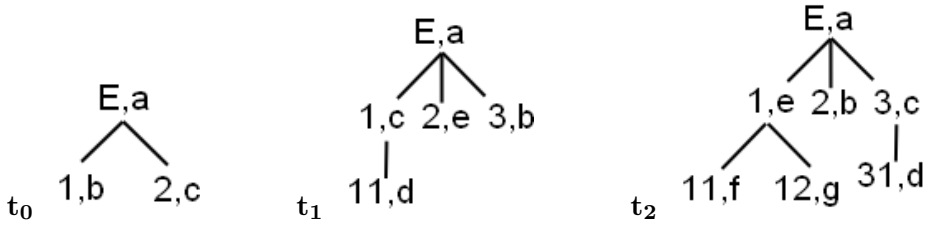
waarbij $\#_c(t_j)$ *gelijk is aan het aantal directe kinderen van de wortel van boom* t_j*.*
$(\overline{t_n})_{n\in\mathbb{N}}$ *is een rij equivalentieklassen van een* **stijgende rij** *eindige bomen. We definiëren de* **limiet** *van* $(\overline{t_n})_{n\in\mathbb{N}}$ *als* $\lim_{n\to\infty} \overline{t_n} = \overline{t}$*, waarbij* t *bepaald is door:*

$$Dom(t) = \{\varepsilon\} \cup \bigcup_{n\in\mathbb{N}} \{\, (a_1 + m_n)a_2a_3\ldots \mid a_1a_2a_3\ldots \in Dom(t_n) \,\} \quad en$$

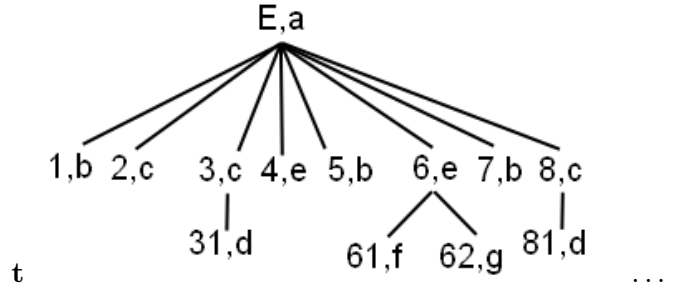
$\forall\alpha \in Dom(t)$ *met* $\alpha = a_1a_2a_3\ldots, \exists i \in \mathbb{N}$ *met* $m_i < a_1 \leq m_{i+1}$ *en* $t(\alpha) = t_i((a_1 - m_i)a_2a_3\ldots)$.

Merk op dat de limiet gelijk gesteld is aan de equivalentieklasse van boom t en we verder boom t zelf beschreven hebben. Analoog hebben we t_n, t_i en t_j beschouwd in plaats van hun equivalentieklassen.

We illustreren deze definitie met een voorbeeld. In figuur 4.4 zijn een aantal bomen getekend, waarmee we volgende stijgende rij kunnen construeren $\overline{t_0}$, $\overline{t_1}$, $\overline{t_2}, \ldots$ Inderdaad we zien dat $\overline{t_0} \subseteq \overline{t_1} \subseteq \overline{t_2}$. Bij elke knoop staat een aanduiding van achtereenvolgens het overeenkomstige element uit het domein en het label, gescheiden door een komma. In plaats van symbool ε, voor de lege string, is een hoofdletter E gebruikt. De boom t uit figuur 4.5 vormt de limiet van de rij $(\overline{t_n})_{n\in\mathbb{N}} = \overline{t_0}, \overline{t_1}, \overline{t_2}, \ldots$ Deze boom is uiteraard verre van volledig gezien enkel knopen uit de drie gegeven bomen opgenomen zijn. Dit is aangeduid met '...' uiterst rechts.



Figuur 4.4: Bomen t_0, t_1 en t_2



Figuur 4.5: Boom t waarvoor het volgende geldt $\lim_{n\to\infty} \overline{t_n} = \overline{t}$

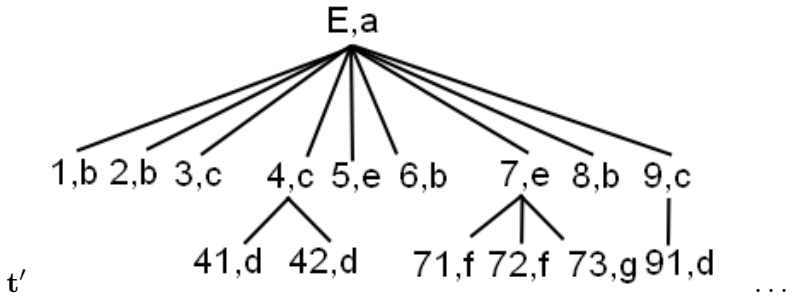
Definitie 4.10 introduceert een variabele m_n. Er zijn geen bomen in de rij voor t_0 dus is $m_0 = 0$. Voor t_1 moeten we het aantal directe kinderen van de wortel van t_0 tellen, dus $m_1 = 2$. En voor t_2 nemen we de som van het aantal directe kinderen van t_0 en t_1 en vinden we $m_2 = 5$. Deze resultaten zijn nodig bij de constructie van de limiet.
Na variabele m_n wordt in de definitie, het domein van t geconstrueerd. t is opgesteld door als wortel een knoop te nemen gelijk aan de wortels van alle bomen in de rij en die te verbinden met alle kinderen van alle bomen in de rij. Dus de eerste twee kinderen van de wortel van t komen uit boom t_0, de volgende drie uit boom t_1, ... Uiteraard kunnen de knopen niet letterlijk overgenomen worden. Zo zien we dat element 1 uit boom t_1 in boom t element 3 geworden is. Dat is logisch, gezien nu de twee directe kinderen van t_0 ervoor komen. En daarvoor komt m_1 van pas. Voor t_1 moeten we $m_1 = 2$ optellen bij het eerste getal van elke string uit $Dom(t_1)$. Analoog moeten we de elementen uit $Dom(t_2)$ aanpassen door $m_2 = 5$ op te tellen bij het eerste getal, ...
Het laatste deel van definitie 4.10 toont ons hoe $t(\alpha)$ gedefinieerd is voor elke string α uit het domein. Hiervoor passen we de techniek met m_i omgekeerd toe. Neem een string β bijvoorbeeld $\beta = 31$. Aan de hand van het eerste getal van de string, getal 3, kunnen we terugvinden uit welke boom t_i deze knoop afkomstig is. We zoeken het getal i waarvoor $m_i < a_1 = 3 \leq m_{i+1}$ waar is. Voor $i = 1$ wordt de vergelijking een feit, dus het label van string 31 in t is te vinden in boom t_1 en het gezochte label is d. Inderdaad het overeenkomstige element in t_1 is $(3 - m_1)1 = 11$. We kunnen eenvoudig verifiëren dat we met het gebruikte interval voor elke knoop in t zijn originele boom in de stijgende rij kunnen terugvinden.

Eigenschap 4.11. *De boom t die in definitie 4.10 geconstrueerd wordt als limiet van een stijgende rij equivalentieklassen voldoet aan definitie 4.1 van een ongeordende boom.*

Bewijs. De vraag is dus of $Dom(t)$ een geldig boomdomein is. Ten eerste moeten we aantonen dat $Dom(t)$ niet leeg is. Dit is een vaststaand feit gezien het altijd minstens ε zal bevatten. Ten tweede moet gelden dat voor alle strings $\alpha, \beta \in \mathbb{N}_0^*$ met $\alpha\beta \in Dom(t)$, ook geldt dat α in $Dom(t)$ zit. Dus stel we hebben een string $\alpha\beta$ in $Dom(t)$ en stel $\alpha = a_1a_2a_3\ldots$, dan bestaat er een getal i waarvoor geldt dat $(a_1 - m_i)a_2a_3\ldots\beta \in Dom(t_i)$. Het domein van t_i is zelf een boomdomein en dus bevat het ook $(a_1 - m_i)a_2a_3\ldots$ Uit de definitie van t weten we dan dat $a_1a_2a_3\ldots = \alpha$ in $Dom(t)$ zit. Ten derde moet het volgende gelden, neem een string $\alpha \in \mathbb{N}_0^*$ en getallen i, j $\in \mathbb{N}$ met $1 \leq i \leq j$ en $\alpha j \in Dom(t)$, daarvoor moeten we aantonen dat ook αi in $Dom(t)$ zit. Stel opnieuw α gelijk aan $a_1a_2a_3\ldots$ Dan bestaat er een k waarvoor geldt dat $(a_1 - m_k)a_2a_3\ldots j \in Dom(t_k)$. Het domein van t_k is zelf een boomdomein dus het bevat $(a_1 - m_k)a_2a_3\ldots i$. Dan bevat automatisch $Dom(t)$ $a_1a_2a_3\ldots i$ of dus αi. □

Eigenschap 4.12. *Definitie 4.1 van* $\lim_{n\to\infty} \overline{t_n} = \overline{t}$ *is niet afhankelijk van de gekozen bomen* t_0, t_1, $t_2, \ldots$ *uit de equivalentieklassen* $\overline{t_0}$, $\overline{t_1}$, $\overline{t_2}, \ldots$

Bewijs. Uit de constructie van t weten we dat t eigenlijk gewoon een wortel is met daaronder naast elkaar de bomen t_0, t_1, $t_2, \ldots$ Neem nu een stijgende rij bomen t'_0, t'_1, $t'_2, \ldots$ waarbij $t'_0 \equiv t_0$, $t'_1 \equiv t_1$, $t'_2 \equiv t_2, \ldots$, met andere woorden er bestaan homomorfismes $h_0 : Dom(t_0) \to Dom(t'_0)$, $h'_0 : Dom(t'_0) \to Dom(t_0)$, $h_1 : Dom(t_1) \to Dom(t'_1)$, $h'_1 : Dom(t'_1) \to Dom(t_1), \ldots$ Voor deze rij kunnen we de limiet $\lim_{n\to\infty} \overline{t'_n} = \overline{t'}$ construeren. We construeren nu een afbeelding h van $Dom(t)$ naar $Dom(t')$ en een afbeelding h' van $Dom(t')$ naar $Dom(t)$. Wanneer we voor deze afbeeldingen kunnen aantonen dat het homomorfismes zijn, dan is de eigenschap bewezen.
We tonen de constructie van afbeelding h eerst op een voorbeeld. Figuur 4.6 toont een mogelijke boom t'. Omdat de knopen in groepjes staan, kunnen we zien op welke bomen t'_0, t'_1, $t'_2, \ldots$ deze limiet t' gebaseerd is. De variabele m_n uit definitie 4.10 wordt voor boom t': $m'_0 = 0$, $m'_1 = 3$, $m'_2 = 6$. Neem bijvoorbeeld element 62 uit $Dom(t)$, een knoop met label g. We weten dat $m_2 = 5$. Boom t_3 is onbekend, maar toch staat vast dat $m_2 < 6 \leq m_3$ waar is. Dus kunnen we berekenen dat $(6 - m_2)2 = 12$ in $Dom(t_2)$ zit. Dan kunnen we het beeld nemen van 12 onder homomorfisme h_2 van $Dom(t_2)$ naar $Dom(t'_2)$, $h_2(12) = 13$. Vervolgens kunnen we berekenen dat gegeven $m'_2 = 6$, $(1 + m'_2)3 = 73$ in $Dom(t')$ zit. We stellen het beeld van de string 62 onder afbeelding h van $Dom(t)$ naar $Dom(t')$ gelijk aan de string 73.



Figuur 4.6: Boom t' waarvoor het volgende geldt $\lim_{n\to\infty} \overline{t'_n} = \overline{t'}$ en $t'_0 \equiv t_0$, $t'_1 \equiv t_1$, $t'_2 \equiv t_2, \ldots$

We zien duidelijk dat de constructie uit het voorbeeld voor elk element uit $Dom(t)$ kan gebeuren en schrijven de constructie van afbeelding h nu formeel uit. Het beeld van ε onder h is ε zelf. Voor elk element $\alpha = a_1a_2a_3 \ldots \in Dom(t)$ zoeken we het natuurlijk getal i, waarvoor geldt dat $m_i < a_1 \leq m_{i+1}$. Daarmee kunnen we de string $\alpha_i = (a_1 - m_i)a_2a_3 \ldots$ opstellen en deze zit in $Dom(t_i)$. Voor de string α_i kunnen we het beeld onder homomorfisme h_i berekenen. Stel $h_i(\alpha_i) = \beta_i$, waarbij $\beta_i = b_1b_2b_3 \ldots$ dus een element is van $Dom(t'_i)$. Nu kunnen we de string $\beta = (b_1 + m'_i)b_2b_3 \ldots$ construeren en β zit in $Dom(t')$. Figuur 4.7 toont een schets van deze constructie. Links staan de transities tussen de domeinen en rechts tussen de geconstrueerde strings.

$$\begin{array}{ccccc} Dom(t_i) & \stackrel{h_i}{\rightarrow} & Dom(t_i') & \qquad \alpha_i = (a_1 - m_i)a_2a_3\ldots & \stackrel{h_i}{\rightarrow} \quad \beta_i = b_1b_2b_3\ldots \\ \text{def. } t\uparrow & & \downarrow \text{def. } t' & \qquad \text{def. } t\uparrow & \qquad\qquad \downarrow \text{def. } t' \\ Dom(t) & \stackrel{\mathbf{h}}{\rightarrow} & Dom(t') & \qquad \alpha = a_1a_2a_3\ldots & \stackrel{\mathbf{h}}{\rightarrow} \quad \beta = (b_1 + m_i')b_2b_3\ldots \end{array}$$

Figuur 4.7: Schets van de constructie van afbeelding h

Voor afbeelding h gebruiken we enkel de bijectieve afbeelding tussen $Dom(t)$ en $Dom(t_n)$ uit definitie 4.10 en de homomorfismes h_i, dus is h logischerwijs zelf een homomorfisme. Op volledig analoge wijze kunnen we een afbeelding h' construeren van $Dom(t')$ naar $Dom(t)$. Ook die afbeelding zal een homomorfisme zijn. Met andere woorden tussen de bomen t en t' bestaan homomorfismes in de twee richtingen en dus zijn het twee equivalente bomen. □

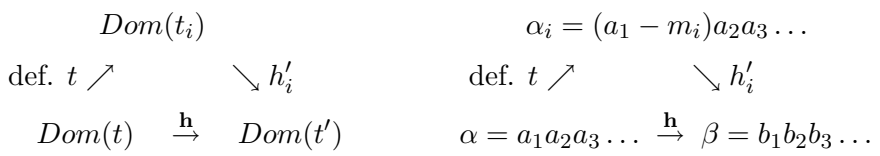
Stelling 4.13. *De limiet van een stijgende rij* $\lim_{n\to\infty} \overline{t_n} = \overline{t}$*, uit definitie 4.10, is de kleinste bovengrens van die rij (op equivalentie na).*

Bewijs. Opdat t de kleinste bovengrens is van de rij moet aan twee voorwaarden voldaan zijn: (i) $\forall i \in \mathbb{N}$: $\overline{t_i} \leq \overline{t}$ (ii) voor elke andere bovengrens $\overline{t'}$ geldt $\overline{t} \leq \overline{t'}$.

(i) Neem een willekeurig getal i. $\overline{t_i} \leq \overline{t}$ als $t_i \leq t$ en hiervoor moet een homomorfisme bestaan van t_i naar t. Uit de definitie van $\lim_{n\to\infty} \overline{t_n} = \overline{t}$ volgt dat elk element $\alpha = a_1a_2a_3\ldots$ uit $Dom(t_i)$ afgebeeld wordt op een element $(a_1 + m_i)a_2a_3$ in $Dom(t)$. We geven deze afbeelding de naam h_i. Als we aantonen dat h_i een homomorfisme is, dan is dit deel bewezen.
h_i is een afbeelding van $Dom(t_i)$ naar $Dom(t)$. h_i voert enkel wijzigingen uit op strings van minstens lengte 1, dus $h_i(\varepsilon) = \varepsilon$. Hiermee is aan de eerste twee condities voor een homomorfisme voldaan. Als derde moeten we checken dat voor alle strings α uit $Dom(t_i)$ geldt dat als $t_i(\alpha) \neq \Omega$ dan $t_i(\alpha)$ gelijk is aan $t(h_i(\alpha))$. Neem een α uit $Dom(t_i)$ en stel $\alpha = a_1a_2a_3\ldots$ Het beeld van α onder h_i is gelijk aan $(a_1 + m_i)a_2a_3\ldots$ en deze string is een element van $Dom(t)$. Met andere woorden, we kunnen $t((a_1 + m_i)a_2a_3\ldots)$ berekenen en dit is volgens de definitie van t gelijk aan $t_i((a_1 + m_i - m_i)a_2a_3\ldots)$. Dus $t(h_i(\alpha)) = t_i(\alpha)$. Als laatste moeten we controleren dat voor alle strings α uit $Dom(t_i)$ en voor alle natuurlijke getallen i met αi in $Dom(t_i)$, geldt dat er een natuurlijk getal j bestaat waarvoor geldt dat $h_i(\alpha i) = h_i(\alpha)j$. Neem dus een bepaalde α en een getal i. Het beeld van $\alpha i = a_1a_2a_3\ldots i$ onder h_i is $(a_1 + m_i)a_2a_3\ldots i$ en het beeld van α is $(a_1 + m_i)a_2a_3\ldots$ We kunnen j dus gewoon gelijkstellen aan i.

(ii) Neem een willekeurige boom t', waarvoor de equivalentieklasse $\overline{t'}$ een bovengrens is van de rij $(\overline{t_n})_{n\in\mathbb{N}}$. Met andere woorden voor elke boom t_i uit de rij geldt $\overline{t_i} \leq \overline{t'}$, dus

geldt $t_i \leq t'$ en bestaat er een homomorfisme h'_i van $Dom(t_i)$ naar $Dom(t')$. We moeten aantonen dat $\bar{t} \leq \overline{t'}$. Dit geldt als $t \leq t'$ en dus als een homomorfisme bestaat van $Dom(t)$ naar $Dom(t')$. De constructie van een afbeelding h gebeurt gelijkaardig als in het bewijs van eigenschap 4.12. In figuur 4.8 is een schets gegeven.



Figuur 4.8: Schets van de constructie van afbeelding h

ε wordt door h op zichzelf afgebeeld. Voor elk element $\alpha = a_1a_2a_3 \ldots \in Dom(t)$ zoeken we het natuurlijk getal i, waarvoor geldt dat $m_i < a_1 \leq m_{i+1}$. Aan de hand daarvan kunnen we de string $\alpha_i = (a_1 - m_i)a_2a_3 \ldots$ opstellen en deze zit in $Dom(t_i)$. Voor de string α_i kunnen we het beeld onder homomorfisme h'_i berekenen. Stel $h'_i(\alpha_i) = \beta$, waarbij β dus een element is van $Dom(t')$. β is het beeld van α onder afbeelding h. Voor afbeelding h gebruiken we opnieuw enkel de bijectieve afbeelding (van elementen uit $Dom(t_i)$ op elementen uit $Dom(t)$) uit definitie 4.10 en het homomorfisme h_i, dus is h zelf een homomorfisme. □

Deze stelling en definitie 3.5 (van compleetheid) leiden tot het volgende besluit:

Gevolg 4.14. *$\leq$ gedefinieerd op de geïntroduceerde equivalentieklassen is een complete partiële orde.*

4.3 Oneindige boom, limiet van rij eindige, ongeordende bomen

We kunnen definitie 3.10 (van bij de geordende bomen) hergebruiken. Deze is hieronder herhaald in definitie 4.15.

Definitie 4.15. *Gegeven een willekeurige boom $t \in M^{\infty}_{\Omega}(F)$ en een natuurlijk getal $n \geq 0$. Dan definiëren we de* **boom t beperkt tot n**, *genoteerd als $t|_n$, als volgt:*
$Dom(\, t|_n\,) = \{\varepsilon\} \cup \{\, \alpha \in Dom(t) \mid \alpha$ *bestaat uit maximum n getallen uit het interval* $[1, n]\,\}$

$$en \quad \forall \alpha \in Dom(t|_n) : t|_n(\alpha) = \begin{cases} t(\alpha) & als \ \ |\alpha| < n \\ \Omega & als \ \ |\alpha| = n \end{cases}$$

Voor $t|_n$ is reeds bewezen dat het een geldig boomdomein is.

Eigenschap 4.16. $\overline{t|_0}$, $\overline{t|_1}$, $\overline{t|_2}, \ldots$ *is een stijgende rij. Met andere woorden* $\forall n \in \mathbb{N} : \overline{t|_n} \leq \overline{t|_{n+1}}$.

Bewijs. Neem een willekeurige $n \geq 0$. Hiervoor moeten we aantonen dat $\overline{t|_n} \leq \overline{t|_{n+1}}$. Dus moeten we aantonen dat $t|_n \leq t|_{n+1}$ geldt, waarvoor er een homomorfisme h moet bestaan van $t|_n$ naar $t|_{n+1}$. Uit definitie 4.15 (van $t|_n$) volgt onmiddelijk dat $Dom(t|_n) \subseteq Dom(t|_{n+1})$. Stel, we nemen als afbeelding h de identiteitsrelatie, dus we beelden elk element van $Dom(t|_n)$ af op hetzelfde element in $Dom(t|_{n+1})$.
Aan de eerste twee voorwaarden voor een homomorfisme (definitie 4.2) is triviaal voldaan. Neem een string α uit $Dom(t|_n)$ met $t|_n(\alpha) \neq \Omega$. Voor α moet gelden dat $t|_n(\alpha) = t|_{n+1}(h(\alpha))$, gegeven dat $h(\alpha) = \alpha$. Omdat $t|_n(\alpha)$ verschilt van Ω volgt uit de definitie dat het gelijk is aan $t(\alpha)$ en dat de lengte van α hoogstens n is. Definitie 4.15 (van $t|_n$) toegepast op t en $n+1$, leidt tot de conclusie dat $t|_{n+1}(\alpha)$ ook gelijk moet zijn aan $t(\alpha)$, omdat $|\alpha| < n$. Dan rest ons enkel nog de controle van de laatste voorwaarde, namelijk voor een bepaalde string in $Dom(t|_n)$ en een natuurlijk getal i met $\alpha i \in Dom(t|_n)$, moet er een natuurlijk getal j bestaan zodat $h(\alpha i) = h(\alpha)j$. We kunnen opnieuw i gewoon gelijkstellen aan j. □

In lemma 3.13 (bij geordende bomen) hebben we reeds aangetoond dat $t|_i$ een eindige boom is, voor elke $i \in \mathbb{N}$.

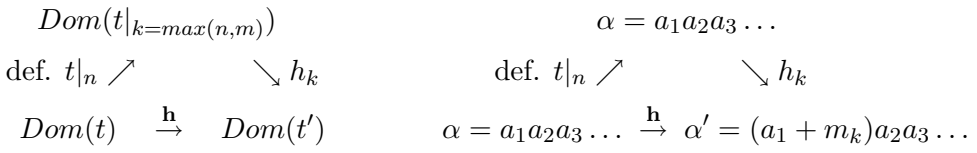
Stelling 4.17. *Elke oneindige boom is de limiet van een rij eindige bomen.*

Bewijs. Neem een willekeurige oneindige boom t en zijn equivalentieklasse $\bar{t}$. Voor t kunnen we de stijgende rij $\overline{t|_0}$, $\overline{t|_1}$, $\overline{t|_2}, \ldots$ construeren. Uit lemma 3.13 weten we dat elke boom $t|_i$ uit die rij eindig is. Elke stijgende rij eindige bomen heeft een kleinste bovengrens (zie vorige paragraaf) en volgens stelling 4.13 is deze gelijk aan de limiet van de rij uit definitie 4.10.
De limiet van de stijgende rij $(\overline{t|_n})_{n\in\mathbb{N}}$ is $\lim_{n\to\infty} \overline{t|_n} = \overline{t'}$, waarbij t' bepaald is door:

$$Dom(t') = \{\varepsilon\} \cup \bigcup_{n\in\mathbb{N}} \{\ (a_1 + m_n)a_2a_3\ldots \mid\ a_1a_2a_3\ldots \in Dom(t|_n)\ \} \quad \text{en}$$

$\forall \alpha \in Dom(t')$ met $\alpha = a_1a_2a_3\ldots, \exists i \in \mathbb{N}$ met $m_i < a_1 \leq m_{i+1}$ en $t'(\alpha) = t|_i((a_1 - m_i)a_2a_3\ldots)$.

Herinner u volgende vergelijking $m_n = \sum_{j<n} \#_c(t_j)$, waarbij $\#_c(t_j)$ gelijk is aan het aantal directe kinderen van de wortel van boom t_j.
Wanneer we nu aantonen dat t en t' equivalent zijn, is de stelling bewezen. t en t' zijn equivalente bomen als $t \leq t'$ en $t' \leq t$ geldt. En daarvoor moet een homomorfisme h bestaan van $Dom(t)$ naar $Dom(t')$ en een homomorfisme h' van $Dom(t')$ naar $Dom(t)$. De constructie van h en h' gebeurt weer gelijkaardig als in het bewijs van eigenschap 4.12. In figuur 4.9 is een schets gegeven voor h.



Figuur 4.9: Schets van de constructie van afbeelding h

ε wordt op zichzelf afgebeeld. Neem een willekeurig element $\alpha = a_1a_2a_3\ldots \in Dom(t)$. Stel de lengte van α gelijk aan n en stel het grootste getal uit de string α gelijk aan m. Stel een nieuwe variabele k gelijk aan $max(n, m)$. We weten dan uit de definitie van $t|_n$ dat α een element is van $Dom(t|_k)$. Nu kunnen we $h_k(\alpha)$ berekenen. $h_k(\alpha) = (a_1 + m_k)a_2a_3\ldots$ Noem deze string α'. Van α' weten we dat het een element is van $Dom(t')$. α' is het beeld van α onder onze afbeelding h.
De constructie van h' is volledig analoog. Voor afbeelding h en h' gebruiken we enkel de bijectieve afbeelding uit definitie 4.15 van $t|_n$ en de homomorfismes h_k respectievelijk h'_k, dus h en h' zijn zelf homomorfismes. □

4.4 Afwerking bewijs gevolg 2.9

Om het bewijs van gevolg 2.9 te vervolledigen maken we gebruik van volgende bewering.

Lemma 4.18. *In een oneindige herschrijvingssequentie $I_0 \overset{v_1}{\rightarrow} I_1 \overset{v_2}{\rightarrow} I_2 \rightarrow \ldots$ is voor elk natuurlijk getal i het systeem I_i eindig.*

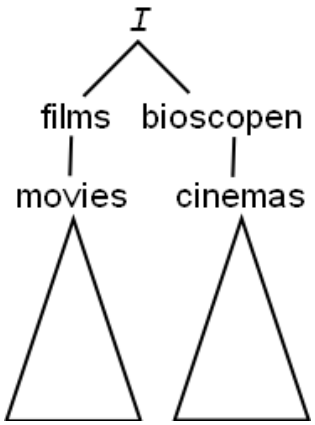
Bewijs. Voor elk AXML systeem I bestaat een expressie (D, F, I), waarbij D een eindige verzameling documentnamen is, F een eindige verzameling functienamen en I een afbeelding over beide verzamelingen. Het ligt voor de hand dat een herschrijvingssequentie begint met een systeem dat enkel eindige documenten bevat. Elke service v_i die uitgevoerd wordt in de sequentie, leidt tot een transitie $I_i \overset{v_i}{\rightarrow} I_{i+1}$ en voegt een eindige hoeveelheid data toe aan de documenten in I_i. Bijgevolg kunnen we besluiten dat het systeem, dat bekomen wordt na een eindig aantal dergelijke transities, nog steeds eindig is. □

Ter herinnering herhalen we hier de te bewijzen bewering.

Gevolg 4.19. *De semantiek van monotone AXML systemen is goed gedefinieerd. Namelijk: (i) Als één herschrijving eindigt, dan eindigt elke herschrijving in hetzelfde eindige systeem. (ii) Als één herschrijving niet eindigt, dan eindigt geen enkele herschrijving en elke rechtvaardige herschrijving produceert hetzelfde oneindige systeem.*

Bewijs. Deel (i) is reeds bewezen in hoofdstuk 2.
Neem een bepaalde oneindige herschrijving $I_0 \xrightarrow{v_1} I_1 \xrightarrow{v_2} I_2 \rightarrow \ldots$ Voor elk natuurlijk getal i is het systeem I_i eindig (lemma 4.18). We weten dat elk zulk systeem beschikt over een eindige verzameling documenten D, waarbij elk document kan voorgesteld worden als een ongeordende AXML boom. Neem een systeem I_0 met D_0 de verzameling met alle documenten. Laten we daarvoor een boom $\mathcal{I}_0$ construeren (zie figuur 4.10). We creëeren een wortelknoop met als label I en voor elk document $d \in D_0$ verbinden we onze wortelknoop met een knoop, gelabeld met d, en deze knoop verbinden we met de wortel van de boom die document d voorstelt. Zo zal dus boom $\mathcal{I}_0$ elk document uit D_0 bevatten.

We illustreren deze constructie op een voorbeeld. Stel we hebben een systeem I met verzameling D gelijk aan {films, bioscopen} en stel dat de overeenkomstige bomen als label voor de wortel respectievelijk *movies* en *cinemas* hebben. Figuur 4.10 toont de structuur van de geconstrueerde boom $\mathcal{I}_0$.



Figuur 4.10: De geconstrueerde boom $\mathcal{I}_0$ voor het gegeven voorbeeld.

Wanneer we nu beginnen aan de herschrijvingssequentie door de functie v_1 uit te voeren. Dan zal boom $\mathcal{I}_0$ aangepast worden gezien het resultaat van de functie toegevoegd wordt. De boom die daardoor bekomen wordt, noemen we $\mathcal{I}_1$. Door het uitvoeren van v_2 bekomen we de boom $\mathcal{I}_2$, ... Gezien enkel met monotone functies gewerkt wordt (zie paragraaf 2.1 met het gekozen model) bekomen we hiermee een stijgende rij bomen $\mathcal{I}_0, \mathcal{I}_1, \mathcal{I}_2, \ldots$ Er is gegeven dat elk systeem I_i eindig is, dus zal elk document uit I_i eindig zijn en zo dus ook de boom die dat systeem voorstelt. Nu weten we uit de vorige paragrafen dat elke stijgende rij eindige, ongeordende bomen een kleinste bovengrens heeft, namelijk de limiet van die rij. En we weten dat de volgorde van functieaanroepen niet uitmaakt (zie paragraaf 2.1), dus zal elke mogelijke herschrijving dezelfde bovengrens bereiken. □

Hoofdstuk 5

Positive Active XML en queries

We gaan nu verder met het bestuderen van het paper Abiteboul *et al.* (2004c) en beginnen bij sectie 3 met als titel Positive Active XML.

In hoofdstuk 2 hebben we monotone AXML systemen gedefinieerd (definitie 2.5). In dit hoofdstuk bespreken we een speciale klasse daarvan, namelijk monotone positive AXML systemen. Deze verschillen van gewone systemen doordat zij services gebruiken, die gedefinieerd zijn door queries waarvan de definitie bekend is. Deze queries noemt men positive queries. Dus in tegenstelling tot de black-box-semantiek die in hoofdstuk 2 gebruikt wordt (zie paragraaf 2.1), weten we hier van elke service welke acties die uitvoert. Merk op dat reeds in het inleidende hoofdstuk aandacht besteed is aan Positive AXML (zie paragraaf 1.6).

In dit hoofdstuk bestuderen we een aantal belangrijke aspecten in verband met queries over AXML documenten. Dit hoofdstuk wijkt bewust af van de exacte, formele uiteenzettingen van vorige hoofdstukken, mede omdat het paper hieromtrent eerder vaag is.

5.1 Positive queries

Positive (AXML) queries komen overeen met een monotoon conjunctief fragment van XQuery (W3C, 2005). Het zijn regels van de vorm *hoofd :- body*. De body voert een selectie uit analoog aan de *from* en *where* gedeeltes uit XQuery en het bevat boompatronen. We proberen voor die boompatronen een matching te vinden op de input documenten, om zo het resultaat van de query te bekomen. Het hoofd van de query komt overeen met het *return* gedeelte van XQuery en dit bestaat uit een boompatroon dat de structuur van het resultaat beschrijft.

Positive queries mogen vier verschillende soorten variabelen bevatten: label-, functie-, waarde- en boomvariabelen. De eerste drie soorten zijn gelijk aan de drie soorten knopen die in een AXML boom (een voorstelling van een AXML document) mogen voorkomen, namelijk knopen gemarkeerd met labels, functienamen of atomaire waardes. In definitie 2.1, de definitie van een

AXML document, zien we dat hiervoor drie verzamelingen gebruikt worden, respectievelijk $\mathcal{L}$, $\mathcal{F}$ en $\mathcal{V}$. De laatste soort variabele in positive queries, een boomvariabele, wordt gebruikt om een deelboom van een document voor te stellen.

Boompatronen zijn duidelijk de belangrijkste component van positive queries. Het zijn deelbomen van AXML documenten waarbij sommige knooplabels vervangen zijn door labelvariabelen, sommige functienamen door functievariabelen en sommige atomaire waardes door waardevariabelen óf boomvariabelen. Herinner u dat in een AXML boom atomaire waardes enkel toegekend mogen worden aan de bladeren.

In definitie 2.5 hebben we een monotoon AXML systeem gedefinieerd als een expressie $(D,\ F,\ I)$. Als de verzamelingen D en F duidelijk zijn uit de context, gebruiken we I om het systeem aan te duiden. Nu kunnen we ook $I(d) = t$ afkorten, voor een bepaalde documentnaam d en een boom t, namelijk met d/t. Dit wordt gebruikt in volgende definitie van positive queries.

Definitie 5.1. *Een* **positive query** *is een expressie*

$$r \ :- \ \ d_1/p_1,\ \ldots,\ d_n/p_n,\ e_1,\ \ldots,\ e_m \quad \textit{waarbij}$$

1. *voor $i = 1, \ldots, n$ geldt dat d_i een documentnaam is en r en p_i positive AXML boompatronen zijn;*
2. *elke variabele in r ook voorkomt in een patroon p_i;*
3. *voor $j = 1, \ldots, m$ geldt dat e_j een ongelijkheid is van de vorm $x \neq y$, waarbij x en y label-, functie- of waardevariabelen zijn en geen enkele boomvariabele meer dan één keer voorkomt in de body.*

Om deze definitie te illustreren, herhalen we het voorbeeld uit hoofdstuk 1. Dat is een service die titels zoekt, van vijfsterrenfilms geregisseerd door Quentin Tarantino, in een AXML document met films. De variabele x in de query is een waardevariabele.

```
titels{x} :- films{film{titel{x},
                        regisseur{"Quentin Tarantino"},
                        rating{"*****"}}}
```

In het deel $'e_1,\ \ldots,\ e_m'$ van een positive query zijn enkel ongelijkheden van variabelen toegelaten. Dit is logisch, gezien voor gelijkheden van variabelen geen expressie e_j nodig is. Het volstaat om een variabele te herhalen. Neem bijvoorbeeld volgende service:

```
titels{x} :- films{film{titel{x},
                        regisseur{x}}}
```

Deze zoekt titels van films waarbij de naam van de regisseur gelijk is aan de naam van de film. De kans dat een dergelijke film bestaat is uiteraard zeer klein, maar het toont wel hoe een gelijkheid van variabelen bekomen kan worden.
Merk op dat voor boomvariabelen zowel ongelijkheden als gelijkheden verboden zijn in positive queries. In de ongelijkheden sequentie $'e_1, \ldots, e_m'$ mogen geen boomvariabelen voorkomen. En tevens wordt geëist dat boomvariabelen geen tweemaal in de body van een query mogen voorkomen, op die manier zijn ook gelijkheden onmogelijk.
Simple positive queries zijn queries die helemaal geen boomvariabelen bevatten. Een positive AXML systeem waarin alle functies gedefinieerd zijn door simple positive queries is een simple positive AXML systeem. Deze systemen verbieden dus het kopiëren van deelbomen. De twee opgegeven voorbeelden zijn simple positive queries.

5.2 Resultaat van positive queries

Voor de semantiek van queries over AXML documenten is een uitbreiding nodig van definitie 2.7 waarin de semantiek van een monotoon AXML systeem beschreven wordt. De semantiek van een document $d \in [\, I \,]$, genoteerd als $[\, d \,]$, is gegeven door $[\, I \,]$. Voor een document d dat géén deel is van I maar enkel functies gebruikt uit I, is de semantiek gegeven door deze van systeem I waaraan d toegevoegd wordt. Dit gegeven hebben we nodig voor het definiëren van de semantiek van queries, omdat de resultaten daarvan nieuwe documenten zijn die geen deel uitmaken van I.

Er zijn twee mogelijkheden voor de semantiek van queries. Ten eerste is er het 'snapshot' resultaat van een query, waarbij de query geëvalueerd wordt op basis van een momentopname van het systeem. Er worden dus eerst geen services uitgevoerd. Ten tweede is er het 'volledige' resultaat dat bekomen wordt door de query te evalueren op de instantie van het monotone systeem die overeenkomt met de semantiek van het systeem. Met andere woorden hiervoor worden eerst alle mogelijke services uitgevoerd en pas dan wordt de query beschouwd.

5.2.1 Resultaat op basis van momentopname systeem

Een toekenning van variabelen μ **respecteert types** als het labels, functienamen, atomaire waardes en bomen toekent aan respectievelijk label-, functie-, waarde- en boomvariabelen. Gegeven een bepaald boompatroon p in een positive query, stelt $\mu(p)$ de boom voor die bekomen wordt uit het patroon door elke variabele te vervangen door een passende waarde.

Stel we hebben een bepaald systeem I dat volgende twee documenten d en d' bevat:

```
d/  r{ t{ a{1}, b{ c{2}, d{3} }},          d'/   a{1}
       t{ a{1}, b{ c{3}, e{3} }},
       t{ a{2}, b{ c{2}, f{6} }}}
```

Neem een waardevariabele x en een labelvariabele z. De volgende query zoekt alle labels van de kinderen van b knopen in t knopen die dezelfde waarde hebben voor hun a kind als deze in document d':

```
z :- d'/ a{x}, d/ r{ t{ a{x}, b{z} }}
```

Het resultaat van deze positive query op basis van een momentopname van het systeem is: `{c, d, e}`. In de documenten van I zitten geen services, dus is dit eigenlijk ook het volledige resultaat van de query. Stel dat we de labelvariabele z in de query vervangen door een boomvariabele Z, dan is het resultaat op basis van een momentopname: `{c{2}, d{3}, c{3}, e{3}}`.

Beschouw een positive query

$$q \;=\; r \;:-\; d_1/p_1,\, \ldots,\, d_n/p_n,\, e_1,\, \ldots,\, e_m$$

(q is de naam van de query, r stelt het resultaat voor) en laat I een bepaald monotoon AXML systeem zijn dat de documenten $d_1, \ldots, d_n$ bevat. Het resultaat van q op I op basis van een momentopname, genoteerd als $q(I)$, is het bos bestaande uit alle documenten $\mu(r)$ zodat:

- μ een toekenning van variabelen is die types respecteert en aan de ongelijkheden voldoet
- voor elke $d_i/\ p_i$ expressie in de body geldt dat $\mu(p_i) \subseteq I(d_i)$.

Voor de semantiek van positive queries gebaseerd op een momentopname van het Positive AXML systeem, kunnen volgende resultaten bekomen worden. We beschrijven een strategie om deze beweringen te staven.

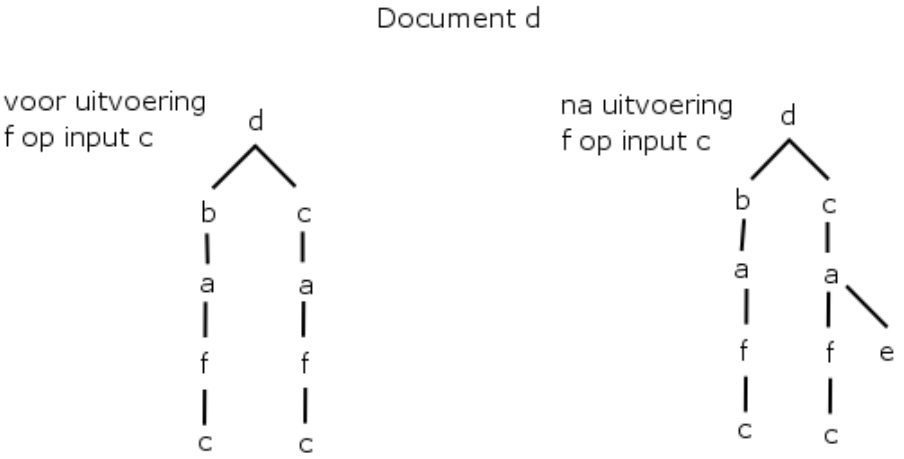
Eigenschap 5.2. *(1) De semantiek van positive queries gebaseerd op een momentopname is monotoon, met andere woorden $I \subseteq J$ impliceert dat $q(I) \subseteq q(J)$. (2) De semantiek is niet monotoon meer indien (on)gelijkheden van boomvariabelen toegelaten zijn. (3) De semantiek kan berekend worden in* PTIME*. (4) Containment van positive queries met deze semantiek is beslisbaar, met andere woorden het is beslisbaar of voor een bepaald systeem I' geldt dat $q(I) \subseteq q(I')$.*

Schets bewijs:

(1) Voor elk positive AXML systeem I bestaat een expressie (D, F, I) met D een eindige verzameling documentnamen, F een eindige verzameling functienamen en I een afbeelding over beide verzamelingen. Voor een systeem J met $I \subseteq J$ zullen de verzamelingen evenveel of meer elementen bevatten als in I. Eventueel bevat J documenten d' waarvoor een document d bestaat met $d \subseteq d'$. Gegeven dat voor het query resultaat enkel variabelen afgebeeld worden

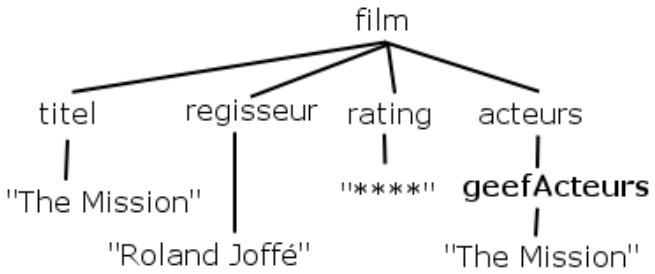
op individuele elementen, is het evident dat de query op J minstens evenveel resultaten zal vinden als op I.

(2) In figuur 5.1 ziet u een document d. Stel we hebben een bepaalde positive query waarin getest wordt op gelijkheid van boomvariabelen: `X :- d/ d{ b{X}, c{X} }`. Hierin is dus X een boomvariabele. We zien in de figuur dat de query vóór de uitvoering van functie f volgend resultaat geeft `a{ f{c} }`. Terwijl de query na het uitvoeren geen resultaat opbrengt. Dit voorbeeld toont aan dat de semantiek van positive queries gebaseerd op een momentopname niet meer monotoon is wanneer gelijkheden van boom variabelen toegelaten worden. Voor ongelijkheden is eveneens een tegenvoorbeeld te vinden.



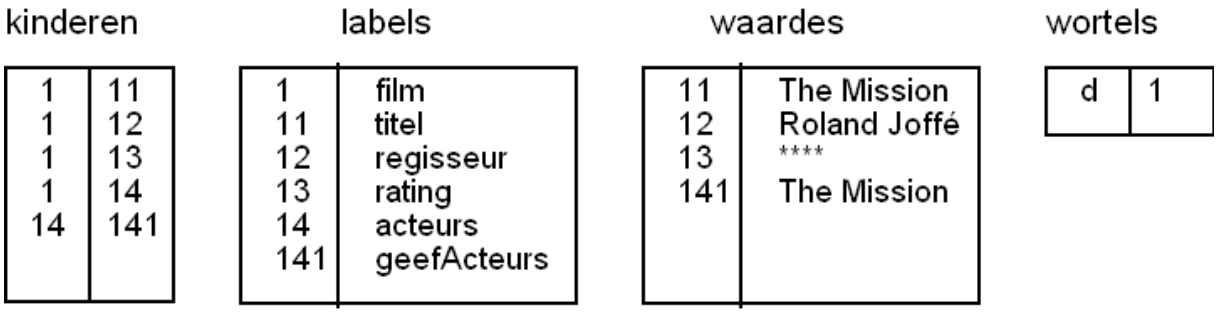
Figuur 5.1: Document d vóór uitvoering van f op input c en na de uitvoering

(3) Volgende strategie kan toegepast worden om het resultaat van positive queries te bepalen op basis van een momentopname van het positive AXML systeem. Sla alle documenten van het systeem op in een relationele database. Functienamen worden beschouwd als gewone labels. Neem bijvoorbeeld document d in figuur 5.2.



Figuur 5.2: Voorbeeld van een Positive AXML document d

We zouden dat document kunnen voorstellen zoals in figuur 5.3. Aan elke knoop is een string van getallen toegekend als unieke id. Merk op dat het in vorige hoofdstukken de gewoonte was om ε te gebruiken voor de wortel van de boom. Omdat we daar nu de string 1 voor gebruiken is onze relatie *wortels* eenvoudiger. Een tweede document in ons systeem zou als wortel 2 kunnen gebruiken. In de relatie *wortels* worden niet enkel de strings van de wortels maar ook de documentnamen bijgehouden.



Figuur 5.3: Relationele database overeenkomstig met document d uit figuur 5.2

Onze positive queries komen overeen met conjunctieve queries, wat een klasse van relationele algebra-expressies is, of met select-project-join expressies en dat is een klasse van SQL queries. We kunnen onze positive queries converteren naar bijvoorbeeld Datalog¬. Hieronder zien we een voorbeeld van een dergelijke vertaling. Uit de cursus *Fundamenten van Databases* weten we dat Datalog¬ vervat is in PTIME.

Een positive query over document d uit figuur 5.2:

```
titels{x} :- film{titel{x},
                  regisseur{x}}}
```

De vertaling hiervan in Datalog¬: (we hebben T, K, L en W gebruikt voor respectievelijk de relaties *titels*, *kinderen*, *labels* en *waardes*)

```
T(t) :- L(f, "film"), K(f,x), K(f,y),
        L(x, "titel"), W(x,t),
        L(y, "regisseur"), W(y,t).
```

(4) Met de strategie uit punt (3) kunnen we zowel voor I als voor I' de semantiek van de query q berekenen. Het resultaat van positive queries is een bepaalde verzameling van strings of representaties van deelbomen. In het voorbeeld hierboven bevat de verzameling titels van films. Gezien positive queries overeenkomen met conjunctieve queries weten we uit de cursus *Databasesystemen* dat containment van positive queries beslisbaar is.

De semantiek van queries die we in deze paragraaf besproken hebben, houdt enkel rekening met de gegevens die op het moment zelf in de documenten van het systeem zitten en negeert alle intensionele data, namelijk de service calls. Dit is niet wat we verwachten bij AXML systemen. Daarom geven we ook een korte beschrijving van een tweede soort semantiek in volgende paragraaf.

5.2.2 Volledig resultaat

Het volledige resultaat van een positive query q voor een monotoon systeem I, beschouwd alle data uit de documenten, zowel de expliciete als de intensionele data. De notatie voor dit resultaat is $[q](I)$.

Als I convergeert naar een eindig systeem $[I]$, dan geldt $[q](I) = q([I])$ (dus het resultaat van q op basis van een momentopname van I). Als we een oneindige, rechtvaardige herschrijvingssequentie $I = I_1 \dots I_i \dots$ bekomen, dan geldt $[q](I) = \cup q(I_i)$.

Op basis van de rechtvaardigheid en monotoniciteit van de herschrijving en de query kan men volgende bewering verifiëren. Opnieuw geven we slechts een schets van het bewijs.

Stelling 5.3. *Het volledige resultaat van een positive query over een monotoon systeem is goed gedefinieerd, met andere woorden het is onafhankelijk van de herschrijvingssequentie.*

We weten uit hoofdstuk 4 dat de semantiek van monotone AXML systemen goed gedefinieerd is. Dus als een systeem I convergeert naar een eindig systeem, dan is er maar één systeem $[I]$ mogelijk. In dat geval is duidelijk dat er voor het volledige resultaat van een query over I ook maar één mogelijkheid is.

Ook voor systemen die leiden tot een oneindige herschrijvingssequentie is de semantiek goed gedefinieerd. In hoofdstuk 4 hebben we een dergelijke sequentie gemodelleerd als een stijgende rij bomen, die een kleinste bovengrens bereikt. Als gevolg daarvan zal het resultaat van query q op die bomen ook steeds groter worden en een bepaalde bovengrens hebben.

Een mogelijke strategie, die in de praktijk toegepast zou kunnen worden, voor het berekenen van het volledig resultaat van een positive query gaat als volgt:

```
Bereken q(I)  (dus op basis van een momentopname)
while ( Gebruiker ontevreden )
     Voer enkele service calls uit
     Bereken q(I)
```

Hoofdstuk 6

Conclusie

In deze thesis zit een grondige introductie tot Active XML op basis van de belangrijkste papers over dat onderwerp. Daarna heb ik, vooral op basis van het paper met als titel Positive Active XML (Abiteboul *et al.*, 2004c), voor alle noodzakelijke begrippen van Active XML een formele definitie gegeven, onder andere voor de semantiek van Active XML systemen. Uit datzelfde paper worden een aantal eigenschappen en stellingen bewezen of een bewijs dat al kort geschetst was, gedetailleerd uitgewerkt. Onder andere wordt aangetoond dat de semantiek van monotone Active XML systemen goed gedefinieerd is. Op die manier tracht ik een bijdrage te leveren tot de theoretische fundering van Active XML.

In mijn laatste hoofdstuk heb ik, op een meer beschrijvende manier, queries over Active XML systemen behandeld. Over dit onderwerp is momenteel reeds een thesis uitgegeven voor het volgende academiejaar, dus zal dit werk voortgezet worden.

Bibliografie

S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo & N. Preda (2004a). Lazy query evaluation for active xml. In *ACM SIGMOD Conference on Management of Data*.

S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo & R. Weber (2002). Active xml: Peer-to-peer data and web services integration. In *Very Large Databases Conference (VLDB), demo*.

S. Abiteboul, O. Benjelloun & T. Milo (2001). Towards a flexible model for data and web services integration. proc. Internat. Workshop on Foundations of Models and Languages for Data and Objects,Italy, 2001.

S. Abiteboul, O. Benjelloun & T. Milo (2004b). Active xml and active query answers. In L. N. in Computer Science, editor, *International Conference on Flexible Query Answer (FQAS)*, volume 3055, pp. 17–27. Springer.

S. Abiteboul, O. Benjelloun & T. Milo (2004c). Positive active xml. In *ACM SIGMOD-SIGACT- SIGART Symposium on Principles of Database Systems*.

Active XML team (2003a). Active XML Primer. `http://activexml.net/`.

Active XML team (2003b). Homepage Active XML. `http://activexml.net/`.

Active XML team (2004). Active XML goes open source. `http://forge.objectweb.org/projects/activexml/`.

Amazon.com (2003). Amazon Web Services. `www.amazon.com/gp/aws/landing.html`.

B. Courcelle (1983). Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169.

T. Milo, S. Abiteboul, B. Amann, O. Benjelloun & F. D. Ngoc (2005). Exchanging intensional xml data. *ACM Transactions on Database Systems*, 30(1):1–40.

OASIS (2000). Universal Description, Discovery, and Integration of Business for the Web (UDDI). `http://www.uddi.org`.

W3C (1996). Extensible Mark up Language (XML). `http://www.w3.org/XML/`.

W3C (1999). XML Path Language (XPath). `http://www.w3.org/TR/xpath`.

W3C (2001a). Web Service Definition Language (WSDL). `http://www.w3.org/TR/wsdl/`.

W3C (2001b). XML Schema. `http://www.w3.org/XML/Schema`.

W3C (2004). Simple Object Access Protocol (SOAP). `http://www.w3.org/TR/soap/`.

W3C (2005). XQuery. `http://www.w3.org/TR/xquery/`.

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen en uw akkoord te verlenen.

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Active XML
Richting: **Master in de informatica** Jaar: **2006**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Deze toekenning van het auteursrecht aan de Universiteit Hasselt houdt in dat ik/wij als auteur de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij kan reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

U bevestigt dat de eindverhandeling uw origineel werk is, en dat u het recht heeft om de rechten te verlenen die in deze overeenkomst worden beschreven. U verklaart tevens dat de eindverhandeling, naar uw weten, het auteursrecht van anderen niet overtreedt.

U verklaart tevens dat u voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen hebt verkregen zodat u deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal u als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze licentie

Ik ga akkoord,

Tamara VOS

Datum: