Over waaiers en grenzen

De constructie van een algoritme voor de optimale ontwikkeling van λ-expressies

Kees Jan van de Looij

Masterscriptie Wijsbegeerte
Faculteit Geesteswetenschappen, Universiteit Utrecht
Departement Wijsbegeerte, Theoretische filosofie
Eerste begeleider: Prof. dr. Albert Visser
Tweede begeleider: Prof. dr. Vincent van Oostrom

augustus 2011

De foto op de titelpagina is gemaakt door Madeleine Hulsen: een close-up van een palm in Kew Gardens.

Inhoud

Voorwoord

Waar deze scriptie over gaat

Als we in een rij symbolen de verschillende instanties van een symbool door een andere rij symbolen moet vervangen, dan kan daarbij dubbel werk ontstaan. Moeten we de nieuwe rijen namelijk later op één en dezelfde manier veranderen, dan hadden we dat beter eerst, vóór het vervangen kunnen doen.

Omdat we liever geen dubbel werk doen, proberen we op een slimme manier symbolen te vervangen, een manier waarop we de symbolen die de plaats van de oorspronkelijke innemen maar één keer op hoeven te schrijven. In de manier waarop we de nieuwe rijen maar één keer representeren speelt de *waaier* een rol.

Hoewel de methoden die dubbel werk vermijden zelf niet heel ingewikkeld zijn, valt het niet mee om te laten zien dat deze methoden ook goed werken, dat wil zeggen, dat ze dezelfde rij symbolen opleveren als wanneer we niet op een slimme manier vervangen, en wel dubbel werk doen.

De λ-rekenkunde is het werkterrein van de methoden die zich richten op het vermijden van dubbel werk. We bekijken er eerst een aantal in vogelvlucht. De rest van de scriptie is een beschrijving van een methode, een beschrijving waar bovendien uit blijkt dat deze methode goed werkt. Naast waaiers heeft die methode ook *grenzen* nodig.

Waar de scriptie ook over gaat

Omdat een λ-expressie een functie of de toepassing van functie op een waarde weer kan geven, gaat de scriptie over wiskunde. De methode voor de optimale ontwikkeling geeft dergelijke expressies alleen op een andere manier weer. We zouden kunnen zeggen dat de methode λ-expressies een andere grammatica geeft.

De scriptie is ook een illustratie voor een bepaalde manier van beschrijven en bewijzen van wiskundige concepten. Omdat ik de methode door de constructie van concepten beschrijf, vind je in de beschrijving niet vaak vervoegingen van 'zijn' terug. In plaats daarvan geef ik aan *wat er kan gebeuren* of *hoe het ene uit het andere volgt*.

Bij de constructie van de concepten in de methode gebruik ik alleen concepten die op voorafgaande concepten berusten. Hoewel dit voor de hand ligt, is het in het geval van een ingewikkelde methode zoals de optimale ontwikkeling van λ-expressies niet zo heel gemakkelijk vol te houden.

Ik begin de constructie van de methode met het vastleggen van de regels voor de interactie

die een boom die de syntax van een λ-expressie weergeeft verandert in een graaf met lokale binding. Deze vorm van binding speelt later een rol bij het vastleggen van wat een optimale stap is: zonder lokale binding hebben we dat soort stappen niet.

Soms ontstaan concepten samen, bijvoorbeeld bij de constructie van het concept van een waaier. We hebben de waaier nodig voor het samenvoegen van takken. Maar een waaier zit optimale stappen in de weg, en voor het verplaatsen van een waaier hebben we niet alleen een regel nodig die de interactie beschrijft, maar ook context.

Wanneer ik een definitie van een voorlopige vorm van context geef, volgt daarmee ook een stukje van het bewijs dat de interactie de expressie die we in een graaf kunnen lezen niet verandert. Want zelfs met die beperkte vorm van context volgt dat de interactie het evenwicht niet verstoort, zodat deze de expressie die we lezen niet verandert.

Zou ik alleen de definitieve vorm van de context vastleggen, dan maakt dat het minder gemakkelijk de constructie van het concept na te lopen. De opvatting van context zou dan namelijk op andere, nog niet uitgewerkte concepten moeten berusten. Veel constructie in de beschrijving, zoals bijvoorbeeld die van een invariant, heeft vaak een voorlopig karakter.

Wanneer we voor de constructie van een concept alleen eerder geaccepteerde concepten gebruiken, volgt een bewijs niet afzonderlijk van de constructie. De beschrijving geeft niet alleen weer *dat* een de methode goed werkt, ze laat ook zien *vanwege welke* concepten ze dat doet. In dergelijke gevallen is wat ik een bewijs noem, een uitspraak zoals ‘door inspectie van de interactieregels’, niet meer dan een mijlpaal.

Een woord van dank

Zonder Vincent van Oostroms idee en uitwerking van een alternatief algoritme voor de optimale ontwikkeling van λ-expressies zou ik me niet hebben kunnen bekwamen in het geven van uitgebreide en veel omvattende bewijzen. Ik denk met plezier terug aan de vele gesprekken die Vincent en ik hebben gehad, gesprekken waarin hij mij kennis heeft laten maken met λ en het idee van een expliciet bereik van abstractie, en met waaiers en grenzen. Zonder zijn hulp zou ik dit terrein niet hebben kunnen overzien, laat staan me erop begeven.

Ook Albert Visser heeft mij enorm geholpen bij het tot een goed einde brengen van dit project. Zijn organisatorische vaardigheid, inventiviteit en overtuigingskracht zijn daarbij van onmiskenbaar belang geweest. Het feit dat Albert een ander perspectief op de materie heeft dan Vincent, heeft ertoe geleid dat ik de eisen die ik aan de uitwerking stelde heb aangepast. Zo heb ik samen met Albert de formalisering van interactienetten uitgewerkt.

Een woord van dank, op een ander niveau

In tegenstelling tot het verhaal over waaiers en grenzen, bestaat er voor jou, Madeleine, geen volgorde, en is niets voorwaardelijk. Dank je dat jij er altijd was, dat je mij gedurende het hele project hebt gevolgd, en tegelijk ook zo lang op mij hebt gewacht. Dank je voor alle praktische ondersteuning, en voor al het advies over grammatica en stijl, maar bovenal voor de kans die je mij geeft om mij vrij te kunnen ontwikkelen tot wie ik graag wil zijn.

Lent, augustus 2011 kj

1 Verschillende antwoorden op dezelfde vraag

Omdat het gunstig is om zo weinig mogelijk werk te doen, dat wil zeggen, zo weinig mogelijk stappen te zetten, vragen we ons af of we één strategie kunnen bedenken die iedere λ-expressie optimaal reduceert. Wanneer we verschillende strategieën uitproberen merken we dat het niet lukt één eenvoudige strategie als de optimale aan te wijzen.

Gelukkig keek Lévy op een andere manier tegen het probleem aan. Hij stelde zich een stap voor die niet één applicatie, maar een verzameling van applicaties in een expressie reduceert. Met het bestaan van een optimale strategie voor dergelijke stappen, hebben we, zij het indirect, ook een optimale strategie voor de reductie van λ-expressies te pakken.

Na een weergave van wat Lévy als optimaal opvat, schetsen we drie algoritmen die stappen zetten die overeenkomen met optimale stappen in de λ-rekenkunde. Bij het bestuderen van die algoritmen krijgen we een indruk van de keuzes die we moeten maken vanwege de interactie die optimale stappen voorbereidt.

Niet alleen optimale stappen zijn van belang. We moeten een nieuw algoritme ook in de bestaande praktijk onderbrengen, dat wil zeggen: het mag niet te veel afwijken van wat we doen wanneer we met λ-expressies werken. Hoewel we de algoritmen niet uitvoerig beschrijven, hier en daar hebben we met losse eindjes te maken, zien we wel op welke manier ze afwijken van de praktijk.

De algoritmen die optimale stappen zetten veranderen grafen. Wanneer we de expressie die een dergelijke graaf weergeeft willen achterhalen, moeten we bij vertakkingen een 'orakel' raadplegen. Na een abstract perspectief op veranderingen in grafen, gaan we in onderdeel 3 verder met een alternatieve bespreking van een algoritme. Alternatief, omdat we geen betekenis aan het orakel geven die buiten die grafen of de interpretatie daarvan ligt.

Lévy's vraag

Bij het wegwerken van applicaties in een λ-expressie, dat wil zeggen, het substitueren van het argument van een applicatie voor de variabele in een abstractie waar de applicatie betrekking op heeft, vragen we ons af of we een strategie kunnen beschrijven die iedere λ-expressie in het kleinst mogelijke aantal stappen reduceert. Lévy [9, p.12] stelt de vraag op de volgende manier.

> Le problème est simple à énoncer. Existe-t-il une stratégie simple pour réduire tout expression M, qui a une forme normale, jusqu'à sa forme normale N en un coût optimal, c'est-à-dire avec un nombre minimal de conversions élémentaires ? Si une telle stratégie existe, peut-on fournir un algorithme correspondant pour évaluer les

λ-expressions de manière optimale ?

Lévy interpreteert een reductie alleen dan als optimaal als deze geen onnodig werk met zich meebrengt. Met werk bedoelen we stappen in de reductie van een λ-expressie. We doen onnodig werk als we een deel van een expressie reduceren dat later in de reductie helemaal wegvalt.

Kijk bijvoorbeeld naar de reductie van

$$(\lambda x.y)((\lambda x.x)(\lambda x.x))$$

Het heeft niet veel zin eerst

$$(\lambda x.x)(\lambda x.x)$$

te reduceren. Want bij de reductie van

$$(\lambda x.y)(\lambda x.x)$$

verdwijnt het deel van de expressie dat je eerder reduceerde: alleen y blijft over.

Lévy ervaart het evenmin als gunstig dat het aantal instanties van een applicatie met een abstractie toeneemt. Zo bevat

$$(\lambda x.xx)((\lambda x.x)(\lambda x.x))$$

één instantie van de expressie

$$(\lambda x.x)(\lambda x.x)$$

en wanneer je de expressie echter reduceert tot

$$((\lambda x.x)(\lambda x.x))((\lambda x.x)(\lambda x.x))$$

komt er nog een instantie bij. De reductie veroorzaakt dubbel werk, want we moeten later beide instanties van de expressie wegwerken. Omdat we in dit geval niet de eerste applicatie, maar juist een volgende applicatie weg moeten werken, heeft het geen zin te proberen dubbel werk te vermijden door altijd eerst de eerste applicatie in de boom weg te werken.

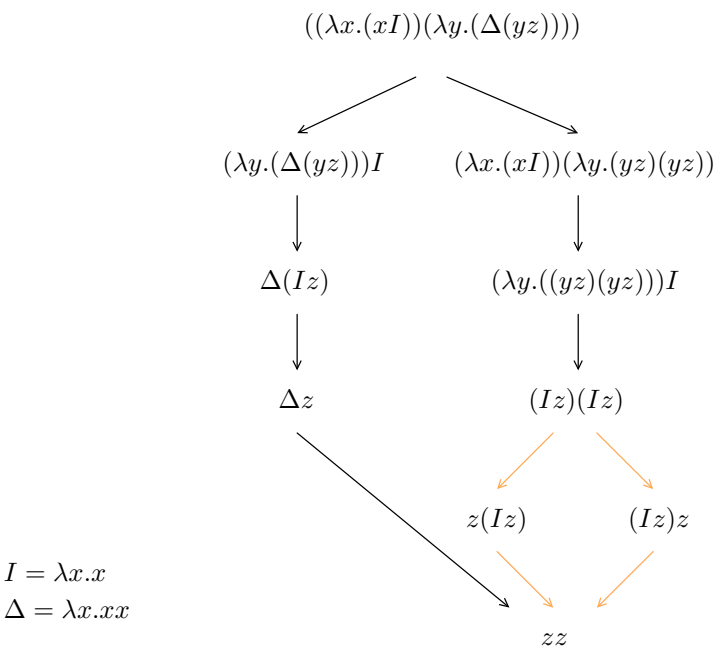
Omgekeerd, als we ons laten leiden door niet de eerste applicatie weg te werken, maar een applicatie die we verderop in de boom tegenkomen, dan zouden we in het geval van het eerste voorbeeld onnodig werk doen: een optimale strategie is dus ingewikkelder dan een regel die zegt altijd de eerste of juist een andere applicatie weg te werken.

We kijken naar een ander voorbeeld. Neem in het volgende overzicht van stappen voor I

$$(\lambda x.x)$$

en voor Δ

$$(\lambda x.xx)$$



$I = \lambda x.x$
$\Delta = \lambda x.xx$

(1)

Als we de stappen aan de rechterzijde van het overzicht zetten, moeten we na de derde stap een keuze maken. Maar onafhankelijk van de stap die we kiezen werken we altijd twee keer dezelfde applicatie weg. Dit betekent dat we dubbel werk moeten doen.

Het de ene keer de eerste, en de andere keer juist niet de eerste applicatie wegwerken leidt via de stappen in de linker reductie (1) niet tot onnodig en dubbel werk. We mogen die reductie dus wel als optimaal opvatten.

Maar het lukt echter niet altijd dubbel werk uit de weg te gaan. Zo moeten we in het geval van

$$(\lambda x.xx)((\lambda x.x)(\lambda x.x))$$

de applicatie

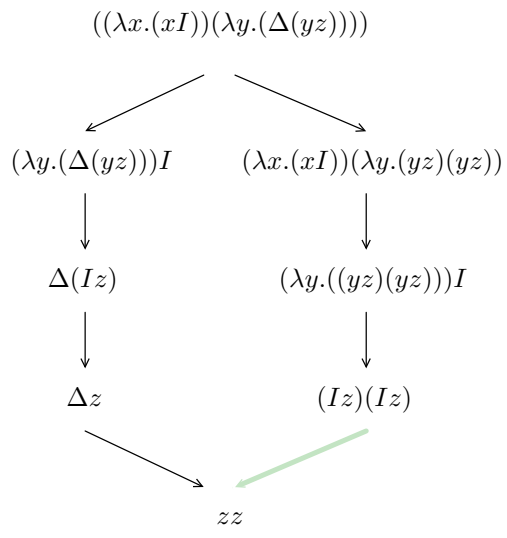
$$((\lambda x.x)(\lambda x.x))$$

altijd twee keer wegwerken. Welke volgorde van stappen we ook kiezen.

Het feit dat je dubbel werk niet kunt vermijden leidt er toe dat we op een andere manier tegen het wegwerken van applicaties aan proberen te kijken. In plaats van applicaties in een expressie stap voor stap te laten verdwijnen, proberen we ons een stap voor te stellen waarmee we meerdere applicaties tegelijk weg kunnen werken.

De stap die Lévy voor ogen heeft werkt alle instanties van een applicatie die met een voorafgaande stap in de expressie verschijnen, weg. Zo verdwijnen de instanties die de

derde stap in de rechter reductie (1) introduceert met één stap, en niet zoals de stappen in de gewone λ- rekenkunde, met twee aparte stappen.



(2)

De verzamelingen die we me de andere stappen (2) wegwerken, bevatten iedere keer maar één instantie van een applicatie. We doen hoe dan ook geen dubbel werk meer.

De stappen zoals Lévy die voor ogen heeft leiden niet alleen tot nieuwe verzamelingen met instanties van een applicatie. De stappen kunnen ook nieuwe instanties van dezelfde applicatie aan de verzameling toevoegen. Als je begint met

$$(\lambda x.xx)((\lambda x.xx)(\lambda x.x))$$

dan ontstaat er met de stap naar

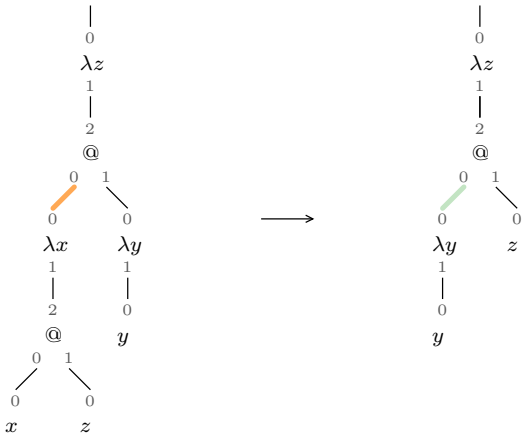
$$(\lambda x.xx)((\lambda x.x)(\lambda x.x))$$

een verzameling met

$$(\lambda x.x)(\lambda x.x)$$

terwijl de volgende stap nog een instantie van deze applicatie aan de verzameling toevoegt.

Naast nieuwe verzamelingen en uitbreidingen van bestaande verzamelingen kan een stap in de reductie van een expressie ook tot een nieuwe applicatie leiden. Met een stap kan een applicatie met een abstractie in de linker tak ontstaan die nog niet in de expressie voorkomt.



(3)

Samen met de y-abstractie die vóór de reductie in de 1-tak van de eerste applicatie in het fragment zit, vormt de tweede applicatie na de reductie een applicatie die we weg kunnen werken. Deze applicatie is geen instantie van een applicatie die we in de expressie lezen; de applicatie komt nog niet in de expressie voor.

Terwijl in de derde stap van de rechter reductie (1, 2) twee instanties van één en dezelfde applicatie ontstaan, kan het wegwerken ook tot verschillende applicaties leiden. Omdat je de applicaties die je samen wegwerkt alleen aan de stap kunt herkennen waarin ze ontstaan, noemen we de verzameling applicaties die met een optimale stap verdwijnen ook wel een familie.

Afgezien van het feit dat ze de applicaties in een expressie in families indelen, weten we nog niet veel van de stappen die tot de optimale reductie van een λ-expressie zouden moeten leiden. Wanneer we naar een dergelijke expressie kijken, zien we in ieder geval niet onmiddellijk welke applicaties van dezelfde familie deel uit maken.

Het ziet ernaar uit dat we, als we op een gemakkelijke manier optimale stappen willen zetten, aan alleen een λ-expressie niet genoeg hebben. Lévy geeft aan [9, p.16] dat als we een andere structuur met de syntaxboom van een expressie in verband brengen, we de optimale stappen ook in die structuur kunnen zetten.

Zodra we het zetten van stappen in een andere structuur als alternatief voor de reductie van λ-expressies zien, mogen de stappen in die structuur vanuit het perspectief van het formele niet, en vanwege het intuïtieve niet te veel afwijken van de stappen waarmee we die expressies reduceren. We kunnen de nieuwe structuur dan namelijk in de bestaande praktijk onderbrengen.

De praktijk van het rekenen met λ-expressies betreft dat wat we met λ-expressies doen: we reduceren ze op een bepaalde manier, en we maken er bomen van die de syntax weergeven, of we betrekken de expressies op een alternatieve rekenkunde. Daarbij zou

je bijvoorbeeld aan de De Bruijn rekenkunde [4] kunnen denken, een rekenkunde die het bereik in een abstractie expliciet weergeeft.

Bij het zoeken naar een alternatieve structuur houden we rekening met de mate waarin we de praktijk van het rekenen met λ-expressies uit moeten breiden. We ervaren een structuur als geschikter naarmate we in minder ingrijpende uitbreidingen van de praktijk moeten voorzien.

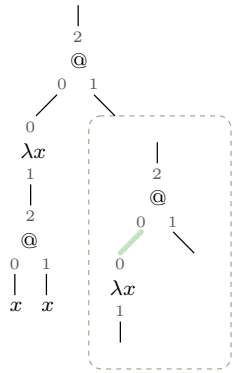
Wanneer we willen laten zien dat de stappen die we in een alternatieve structuur zetten optimaal zijn, hebben we een definitie nodig van de relatie die de verzamelingen van applicaties vastlegt die met die stappen verdwijnen.

Hoewel Lévy drie formele en equivalente definities van de relatie vastlegt, gebruiken we die niet, in ieder geval niet rechtstreeks. Wanneer we laten zien dat de structuur die we vastleggen optimale stappen met zich meebrengt, doen we dat door de stappen te vergelijken met de stappen in een algoritme waarvan we weten dat het optimaal is.

Het algoritme van Lamping

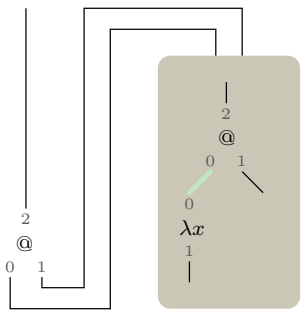
Hoewel Lévy verschillende, equivalente definities van het begrip familie geeft, voorziet hij niet in een algoritme waarmee je optimale stappen kunt zetten. Met Lévy's these begint een zoektocht naar een algoritme dat in plaats van applicatie na applicatie, alle applicaties die deel uit maken van eenzelfde familie met één stap wegwerkt.

Nog voordat Lévy het begrip familie vastlegt, geeft Wadsworth [17] een algoritme dat dubbel werk probeert te voorkomen. Bij een substitutie verbindt dit algoritme namelijk de takken die naar de instanties van de variabele leiden die bij de abstractor hoort met het argument. Zo verandert



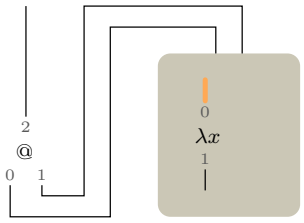
(4)

in



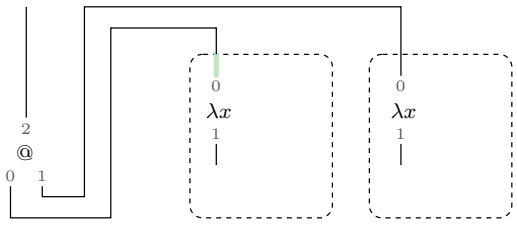
(5)

De applicatie in het argument van de eerste applicatie lees je in beide takken van de eerste applicatie; in iedere tak lees je één instantie. Maar het aantal takken waarin je een instantie leest bepaalt niet hoeveel werk je moet verzetten, want je hoeft alleen maar de applicator en de abstractor weg te halen.



(6)

Wanneer we echter een abstractie in de gemeenschappelijke takken lezen, moeten we deze toch weer op meerdere plaatsen weergeven. Bij het wegwerken van de applicatie moet de abstractor in de 0-tak van de applicator namelijk verdwijnen, terwijl er aan de knopen in de 1-tak niets mag veranderen.

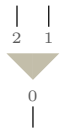


(7)

Maar zodra we de gemeenschappelijke takken op meerdere plaatsen in de graaf terugzien, verliezen we het optimale uit het oog: we geven namelijk applicaties die in dezelfde familie zitten op verschillende plaatsen weer. Het lukt dan niet meer met één stap, een lokale verandering, alle applicaties in de familie weg te werken.

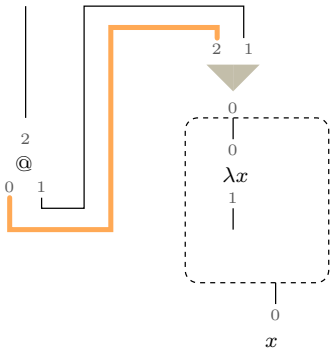
Met de bedoeling grafen wel op een lokale manier te veranderen, stelt Lamping [11] een

knoop voor die de takken paarsgewijs samenvoegt: de waaier.



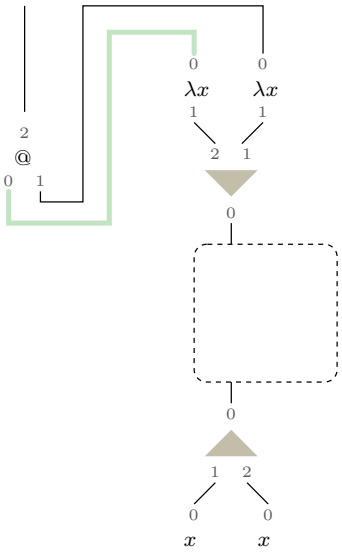
(8)

Als we een waaier gebruiken om de takken van de applicator (6) samen te voegen, krijgen we de volgende graaf.



(9)

Wanneer we meer dan twee takken willen samenvoegen, hebben we meer waaiers nodig: één waaier voor iedere volgende tak. De waaier is niet alleen gunstig omdat ze takken kan samenvoegen. Met een waaier kunnen we ook één knoop uit een gemeenschappelijke tak halen, terwijl we de rest van de knopen in die tak laten zitten.



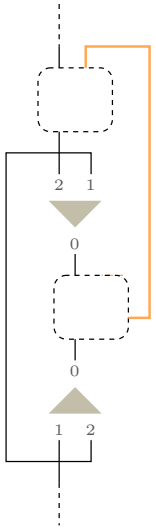
(10)

Omdat we bij het afzonderen twee abstractors krijgen, kunnen we de instanties van de variabele niet langer op één plaats weergeven. Met een waaier splitsen we de tak die naar de enkelvoudige weergave van de instanties leidt in twee aparte voorstellingen van een variabele.

Verdwijnt de abstractor waarmee de 0-tak van de applicator begint, dan neemt de 1-tak van de applicator de plaats van één van de variabelen in de takken van de tweede waaier in. Omdat we in de tweede waaier een tak moeten kiezen, spreken we af dat als we in de richting (0,n) door de eerste waaier lopen, we de richting (n,0) door de tweede waaier gaan.

Met de verandering zonderen we de abstractor van de rest van de abstractie af. Omdat we die nog steeds maar op één plaats in de graaf weergeven, doet de verandering geen afbreuk aan het optimale in de graaf. Verdwijnt de abstractor, dan verdwijnen alle applicaties in een familie; we zetten een optimale stap.

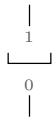
Vanwege het onthouden van de richting horen waaiers bij elkaar. Wanneer het ene paar op het andere volgt, hoeven we nooit meer dan één richting te onthouden. Maar als we na de eerste waaier (10) via een gebogen tak weer in hetzelfde fragment terechtkomen, lopen we voor een tweede keer door de eerste waaier, terwijl we nog niet in de tweede waaier zijn geweest.



(11)

Wanneer we voor de tweede keer door de eerste waaier lopen, moeten we de richting van de eerste gang door de waaier onthouden. Lamping voorziet hiervoor in een stapel met niveaus. Op ieder niveau in de stapel bewaren we de naam van de poort via welke we de eerste waaier binnengaan. Het bovenste niveau bevat de naam die bij de meest recente waaier hoort.

Lamping brengt de niveaus via een nieuwe knoop met de grafen in verband.



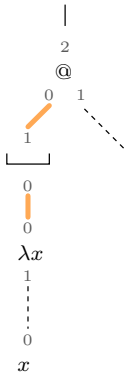
(12)

Bij een openhaakje voegen we een niveau toe, bij het sluithaakje halen we het niveau weer weg.

Als we er voor zorgen dat grafen alleen op een zodanige manier veranderen dat haakjes waaiers die een paar vormen (10) omgeven, dan zou ieder paar waaiers over een eigen niveau beschikken, zodat er geen verwarring kan ontstaan over de tak die we, wanneer we door de tweede waaier in het paar lopen, moeten kiezen. We hoeven dan namelijk alleen maar naar het juiste niveau in de stapel te kijken.

Vanwege haakjes mogen we een tak die in een abstractor begint, en eindigt in een instantie van de variabele die bij de abstractor hoort, transparant opvatten: omdat we aan het einde van de tak dezelfde niveaus zien als aan het begin, lezen we in wat de rechter tak van de applicatie was hetzelfde als vóór de verandering. Die komt daarmee met substitutie overeen.

Als waaiers een instantie van een variabele voorafgaan, en we na het verdwijnen van die variabele na die waaiers verder gaan in een tak waarin in een abstractor zit, terwijl we vóór de waaiers via een applicator lopen, dan zitten die waaiers tussen een applicator en een abstractor in. Omdat haakjes de waaiers omgeven, zitten ook deze haakjes tussen die applicator en abstractor.

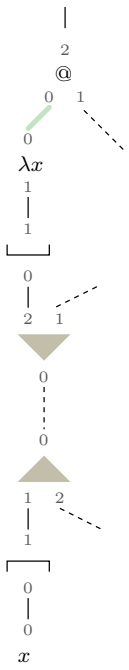


(13)

Wanneer we door de graaf lopen lezen we een expressie waarin een applicatie zit die we weg zouden kunnen werken. Omdat een eenvoudige regel voor het wegwerken van de applicatie alleen vastlegt wat er met de applicator en de abstractor gebeurt, moeten we

eerst een regel voor een verandering geven, zodanig dat het haakje niet meer tussen de applicator en de abstractor in zit.

Zouden we het haakje en de abstractor verwisselen, dan zien we aan het einde van de tak die in de abstractor begint echter een andere stapel dan aan het begin, zodat de graaf niet langer transparant is. We weten zeker dat we de 1-tak van de applicator wel op dezelfde manier lezen als we het niveau dat we aan het begin van de tak toevoegen, aan het einde ook weer weghalen.



(14)

We veranderen het haakje daarom in een voorwaardelijk haakje: in plaats van het niveau weg te halen, verbergen we het alleen. Spreken we af dat we aan het einde van de tak bij een voorwaardelijk haakje dat we in de tegenovergestelde richting passeren, het niveau weer toevoegen, dan begint en eindigt de tak weer met dezelfde stapel, en blijft de graaf transparant.

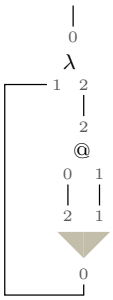
Lamping geeft aan [10, p.29] dat het algoritme niet werkt als we ieder paar waaiers met een paar haakjes omgeven. Wanneer we namelijk in een tak een expressie met een vrije variabele moeten lezen, blijkt dat we alleen de juiste expressie lezen als we extra haakjes aan de graaf toevoegen. Gonthier, Abadi, en Lévy beschrijven [6, p.7] deze haakjes.

Het algoritme van Asperti en Guerrini

De zoektocht naar een algoritme waarmee je een expressie optimaal kunt reduceren ging verder. Het algoritme dat Asperti en Guerrini [1] beschrijven werkt met grafen waarin de takken die naar instanties van de variabele leiden die bij een abstractor hoort, niet in die instanties, maar in de abstractor eindigen. Neem bijvoorbeeld de graaf waarin we

$$\lambda x.xx$$

lezen.



(15)

Ook in het geval waarin er meer dan twee instanties van een variabele in een expressie voorkomen, geven we binding op een lokale manier weer: de tak die naar een derde instantie leidt, voegen we met een extra waaier samen met de tak die in de abstractor eindigt. Vanwege het lokale hoeven we de naam van de variabele die de abstractor bindt niet langer weer te geven.

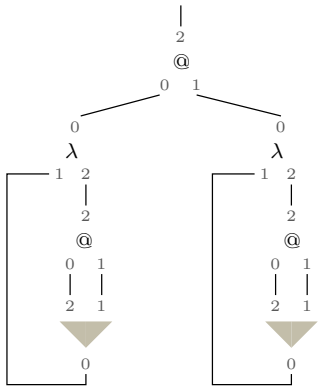
Omdat we de instanties van de variabele op één plaats in de graaf lezen, beperkt het wegwerken van applicaties zich tot een eenvoudige, lokale verandering in de graaf. Als er een applicator aan de abstractor voorafgaat, dan hoeven we de tak die in de abstractor eindigt alleen maar met het argument van de applicator te verbinden. Met lokale binding kunnen we meerdere applicaties met één stap, wegwerken.

Asperti en Guerrini signaleren [1, p.39] het tekort aan haakjes in de grafen waarmee het algoritme van Lamping werkt ook. Ze verbinden de niveaus daarom niet, zoals Lamping dat doet, met een paar waaiers, maar met de applicator. Hoewel de knopen er soms hetzelfde uitzien, krijgen we met deze benadering wel een ander algoritme.

Wanneer we een boom vertalen door de takken die naar instanties van variabelen via waaiers samen te voegen, en in de bijbehorende abstractors te laten eindigen, lezen we de expressie die we verwachten te lezen. Zo lezen we

$$(\lambda x.xx)(\lambda x.xx)$$

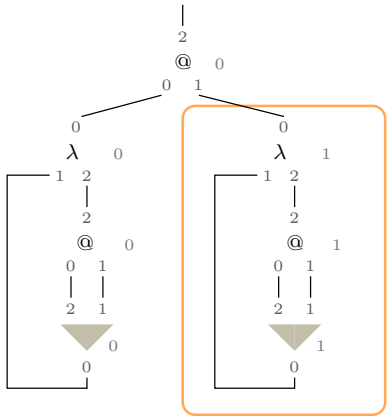
in



(16)

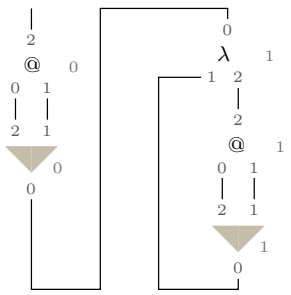
Wanneer we echter de eerste applicatie wegwerken, en het argument daardoor de plaats van de variabele inneemt, gebruiken beide waaiers hetzelfde niveau. Hoewel dat in deze graaf niet voor problemen zorgt, ontstaat er wel verwarring, als meerdere paren met waaiers bij hetzelfde niveau zouden horen.

Asperti en Guerrini kiezen er daarom voor de waaiers in het argument aan een ander niveau te verbinden dan de waaiers in de abstractie. Ze breiden daartoe het label van de knopen uit, zodanig dat de knopen in de 1-tak van een applicator een andere uitbreiding krijgen dan die in de 0-tak. Neemt een argument de plaats van een variabele in, dan wijzen de waaiers in het argument daardoor naar een ander niveau dan die in de abstractie.



(17)

Ondanks de indices kan er bij de reductie van de expressie in de graaf (17) toch weer verwarring ontstaan. Wanneer we namelijk via de 0-tak van de tweede applicator door de waaier lopen zou deze op grond van de manier waarop Asperti en Guerrini de indices toevoegen eigenlijk naar niveau 0 en niet naar 1 moeten wijzen.



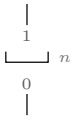
(18)

Wanneer we vóór de knopen in het argument echter een knoop tegen zouden komen die een niveau toevoegt, dan wijzen de knopen die je via de 0-tak van de tweede applicator tegenkomt wel naar het juiste niveau. Asperti en Guerrini voegen daarom bij het vertalen van een expressie, voordat we een variabele zouden lezen, een knoop toe die precies dit effect heeft, de croissant.

(19)

Vanwege de vertaling komen we ook in de 1-tak van de eerste applicator een croissant tegen. Omdat de croissant net als de andere knopen in de 1-tak van de applicator 1 als uitgebreid label heeft, wijst deze niveau 1 in de stapel aan. Als we, wanneer we door de croissant lopen, onder het bovenste niveau een niveau 1 toevoegen, dan hebben de waaiers in de 1-tak een eigen niveau.

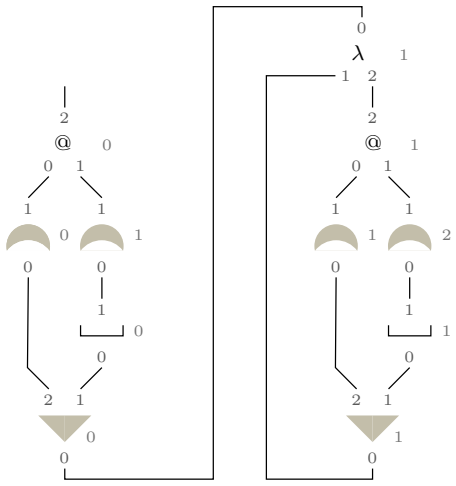
Omdat de knopen in de 1-tak van de eerste applicator en de knopen in de 1-tak van de tweede applicator naar hetzelfde niveau wijzen, hoeven de niveaus wanneer we de 1-tak lezen niet op te schuiven. Asperti voegen daarom een knoop toe die het nieuwe niveau tijdelijk onzichtbaar maakt, het haakje.



(20)

Wanneer we de croissant en het haakje bij het vertalen van een expressie aan de graaf toevoegen krijgen de croissants in de 1-tak van de applicator hetzelfde uitgebreide label als de andere knopen in die tak. Vanwege het niveau waar het haakje effect op moet hebben, krijgt het haakje 0 als uitgebreid label.

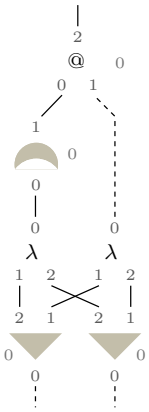
De waaier in de vertaling wijst niet alleen in de 0-tak van de applicator niveau 0 aan: in de 1-tak doet ze dat ook. De waaier mag naar hetzelfde niveau wijzen, omdat ze in de vertaling in de tak van een abstractor zit en we haar daarmee niet zowel in de 0-tak als in de 1-tak van een applicator die aan die abstractor voorafgaat kunnen tegenkomen.



(21)

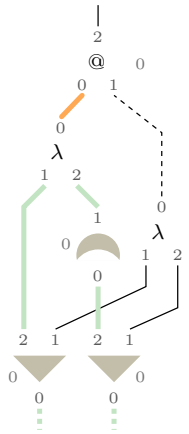
Het wegwerken van de applicatie verandert de uitgebreide labels van de knopen niet. De croissants en de haakjes zorgen er daarbij voor dat er bij het onthouden van namen van poorten geen verwarring ontstaat.

Omdat we in de graaf (21) eerst een applicator, en daarna een abstractor tegenkomen, lezen we een expressie die we verder kunnen reduceren. Vanwege het feit dat we op weg van de applicator naar de applicator echter een waaier tegenkomen, kunnen we niet zonder meer de applicator en de abstractor weglaten. Omdat de abstractor waarin je via de 1-tak van de applicator terechtkomt niet verdwijnt, moeten we de waaier verplaatsen.



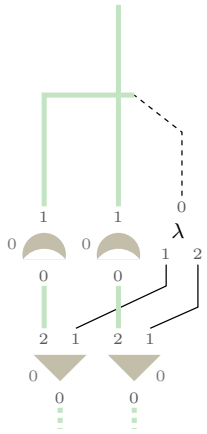
(22)

Daarbij verschijnt er een waaier aan het begin en een waaier aan het einde van de gebogen tak van de abstractor. Na het verdwijnen van de waaier zit de croissant nog tussen de applicator en één van de abstractors in. Zou je de croissant aan de applicator vooraf laten gaan, dan verandert het niveau waar de knopen in de 1-tak naar wijzen. Deze verplaatsing zou de expressie die we lezen veranderen.



(23)

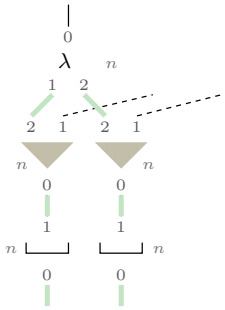
Wanneer we de croissant echter op de abstractor laten volgen, veranderen de niveaus in de 1-tak in eerste instantie niet. Zodra de applicator en de abstractor echter verdwijnen, leidt de aanwezigheid van de croissant ertoe dat we na de gebogen tak van de abstractor met andere niveaus aan de 1-tak van de applicator beginnen.



(24)

Haalt de tweede croissant het niveau weg dat de croissant aan het begin toevoegt, dan hangt het niveau waarmee we aan de 1-tak beginnen alleen af van wat er in de gebogen tak met de niveaus gebeurt. Zou er niets gebeuren, dan volgt dat we na de reductie aan het einde van de gebogen tak met dezelfde niveaus en inhoud aan de 1-tak beginnen.

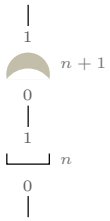
Het verplaatsen van een haakje leidt tot een gelijkaardige overweging. Bij het verplaatsen zien we dat een haakje alleen een niveau mag verbergen: als er na de haakjes nog een paar waaiers in de gebogen tak verschijnt, hebben we het niveau dat we bij het eerste haakje onzichtbaar maken bij de tweede namelijk weer nodig.



(25)

Aan het verdwijnen van de applicator en de abstractor zie je niet onmiddellijk dat we met dezelfde niveaus aan de 1-tak beginnen, of dat we tenminste dezelfde expressie in die tak lezen. Maar Asperti en Guerrini bewijzen dat het algoritme werkt: de verandering in de grafen volgt de reductie van de expressie die de vertaling weergeeft. Ze laten bovendien zien dat we de stap die we zetten wanneer we een applicator en een abstractor wegwerken, als een optimale stap mogen opvatten.

Wat het algoritme van Asperti en Guerrini niet helpt is het feit dat in bepaalde gevallen [1, fig 9.3] de samenstelling van een croissant en een haakje geen merkbaar effect op de niveaus heeft, terwijl de aanwezigheid van deze knopen wel tot verplaatsingen leidt.



(26)

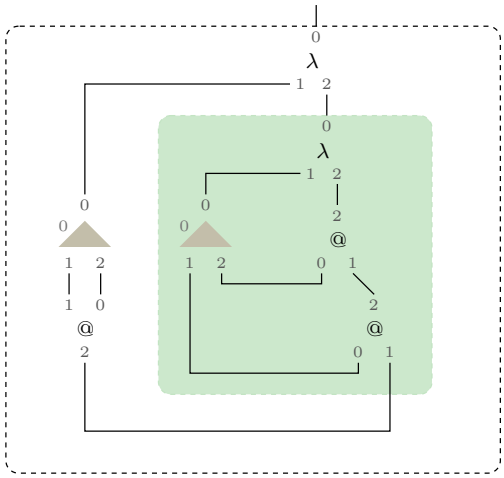
Asperti en Guerrini beperken het probleem van de opeenhopingen door nieuwe regels voor de verandering van grafen aan hun algoritme toe te voegen.

Anders dan het algoritme van Lamping, heeft het algoritme van Asperti en Guerrini geen last van een tekort aan haakjes. Het feit dat dit algoritme zich wat niveaus betreft op applicators, en niet, zoals het algoritme van Lamping dat doet, op waaiers concentreert, maakt het nodige verschil.

Lambdascope

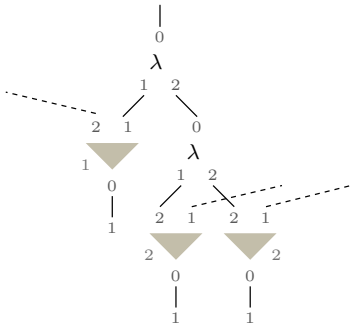
Wanneer we het algoritme van Lamping toepassen, leiden haakjes die waaiers omgeven tot de niveaus waarop we namen van poorten onthouden. In het algoritme van Asperti en Guerrini bepalen de applicators die niveaus. We kunnen echter ook de abstractor met niveaus in verband brengen. Dit gebeurt in Lambdascope, een ander algoritme [16] voor de optimale reductie van λ-expressies.

Zodra er een paar waaiers in de gebogen tak van een abstractor terechtkomt, moeten we namen van poorten onthouden. Omdat het ene paar waaiers in de gebogen tak het andere omvat, hoeven we niet, zoals in het algoritme van Lamping voor ieder paar waaiers een apart niveau toe te voegen: aan één niveau bij iedere abstractor hebben we genoeg.



(27)

Als we de eerste waaier in een paar weglaten, en de tak van de abstractor die op die waaier volgt, met een waaier laten beginnen en eindigen, vormt de eerste waaier in het nieuwe paar een paar met de tweede waaier in het oorspronkelijke paar.

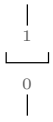


(28)

We onthouden namen van poorten daarom op het niveau dat bij de abstractor hoort waarin de 0-tak van de waaier oorspronkelijk (27) eindigt. Met een uitbreiding van het label laten we een waaier naar die abstractor wijzen. Zo geeft label 1 aan dat de waaier niet bij de voorafgaande abstractor hoort, maar bij de abstractor die daar aan voorafgaat.

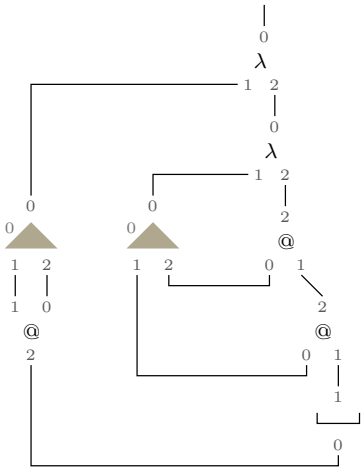
Wanneer we verschillende abstractors lezen, hebben we ook met meer dan één niveau te maken. Omdat we bij de tweede abstractor (27) een niveau toevoegen, wijst de tweede waaier niet meer naar het juiste niveau.

Zouden we weten wanneer we het gebied verlaten waarin de waaiers bij de voorafgaande abstractor horen, dan kunnen we daar het bovenste niveau van de stapel weghalen. Met behulp van een extra knoop geven we de grens van dit gebied aan.



(29)

Zodra we de grens passeren, halen we het niveau dat bij de voorafgaande abstractor hoort weg.

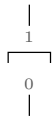


(30)

De grens geeft niet alleen aan dat er geen waaiers meer volgen, ze geeft ook aan we geen instantie van de variabele meer lezen die bij de abstractor hoort. De gebogen takken van een abstractor komen dus overeen met het bereik van die abstractor in de λ-expressie die we in de graaf lezen. Het niveau dat we bij een abstractor toevoegen kunnen we daardoor in verband brengen met bereik. Vandaar de naam van het algoritme.

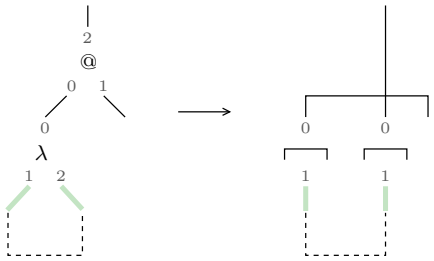
Omdat we bij een abstractor een niveau toevoegen, zou er bij het verdwijnen van een applicator en een abstractor een niveau ontbreken. Dit terwijl er niets met de waaiers in

de tak gebeurt, en we het niveau dus nodig hebben voor het onthouden van poorten. Dit betekent dat we een knoop moeten toevoegen die de rol van de abstractor overneemt: op de plaats van de abstractor voegen we een grens toe die we niet zoals de grens die een aftakking (29) aangeeft in de richting (1,0) maar in de richting (0,1) oversteken:



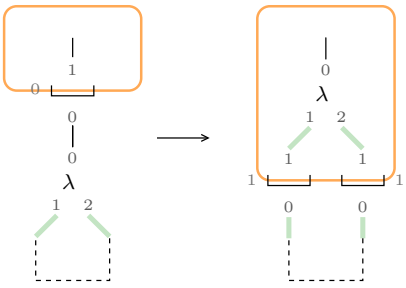
(31)

Wanneer we deze grens oversteken, voegen we een niveau aan de stapel toe. Omdat we na de verandering met dezelfde stapel aan het argument van de applicator moeten beginnen als vóór de verandering, moeten we aan het einde van de gebogen tak het niveau weer van de stapel halen. Het wegwerken van applicaties ziet er in Lambdascope daardoor als volgt uit.



(32)

Ook bij Lambdascope kan een knoop het wegwerken van een applicatie met een abstractie in de weg zitten. Wanneer we een grens achter een abstractor leggen betekent dit niet dat we het gebied verlaten dat bij de voorafgaande abstractor hoort. We verlaten namelijk het gebied dat bij de abstractor hoort die daar weer aan voorafgaat.

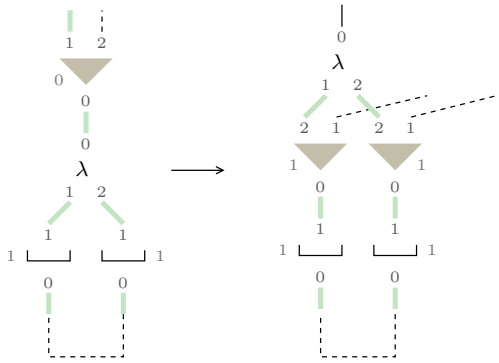


(33)

Breiden we het label van een grens uit met het aantal abstractors dat tussen die grens en de bijbehorende abstractor ligt, dan kan er geen verwarring ontstaan. Want aan de

uitbreiding van het label van de grens zien we dan dat deze niet bij de abstractor hoort die onmiddellijk voorafgaat, maar bij een andere.

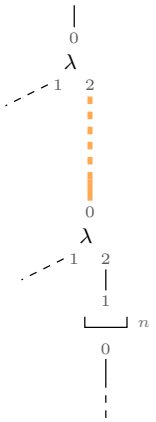
Na het oprekken van de grens (33) overlapt het ene gebied het andere. Dit betekent dat we een waaier de grens kunnen laten volgen: in de tak verschijnt een paar waaiers waarvan de uitbreiding in het label één groter is dan de oorspronkelijke waaiers. Net als de grenzen blijven de waaiers met deze veranderingen met hetzelfde niveau verbonden.



(34)

De waaiers in de tak (33) van de abstractor horen niet bij het niveau dat bij de grens verdwijnt. Zouden we bij de eerste grens (34) het niveau echter weghalen, dan zien we bij de tweede waaier niet langer de juiste poort. We verbergen daarom het niveau bij de eerste grens, en halen het bij de tweede weer tevoorschijn.

Lambdascope heeft geen last van samenstellingen (26) die in bepaalde gevallen geen effect hebben. Want als we onmiddellijk na een abstractor een niveau verbergen, dan doen we dat bij een uitgetrokken grens. Het niveau dat we daarbij verbergen kan daarom niet bij de voorafgaande abstractor horen; we verbergen altijd een ander niveau.



(35)

Lambdascope werkt niet met bomen die de syntax van een expressie weergeven, maar met grafen die het bereik van de abstractors expliciet weergeven. Het algoritme gaat uit van lokale aanduidingen van bereik, dat wil zeggen: het werkt met grafen waarin een knoop het einde van het bereik van de abstractor aangeeft die onmiddellijk voorafgaat.

De rekenkunde die Van Oostrom en Hendriks beschrijven [8] beschikt met λ over een bewerking waarmee je in λ-expressies het einde van het bereik van een abstractor kunt aangeven. Als je iedere λ op de voorafgaande abstractor betrekt die er onmiddellijk aan voorafgaat, maakt dat het vertalen van expressies in grafen gemakkelijk. Het haakje (29) komt dan namelijk overeen met de bewerking die het einde het bereik aangeeft.

Zowel Lambdascope als het algoritme van Asperti en Guerrini voegen bij de vertaling van expressies waaiers aan de graaf toe. Omdat beide algoritmen applicaties met eenzelfde stap wegwerken, volgt met het bewijs van het optimale van het algoritme van Asperti Guerrini [1, p.140] ook het optimale van Lambdascope.

Een andere benadering

Met het doorgronden van het orakel [1, p.281] doelen Asperti en Guerrini op de haakjes en croissants die we nodig hebben wanneer we een graaf interpreteren. Het orakel zorgt ervoor dat we, wanneer we door een waaier lopen, de juiste poort kiezen, en daardoor de juiste expressie lezen.

Over het zoeken naar een betekenis merkt Lamping [11, p.4] echter het volgende op.

> Finally, it is important to note that the notion of paths is part of an explanation of what lambda expression a graph stands for, not something which the algorithm is directly sensitive to. The algorithm simply does local operations, which are in accord with the bigger picture without being directly cognizant of it.

De opmerking geeft aan dat we knopen vanwege het algoritme geen betekenis hoeven te geven die buiten de graaf ligt. Zouden we, zoals de beschrijving [16] van Lambdascope weergeeft, takken in de graaf met het bereik van een abstractor identificeren, dan leggen we echter wel het soort verband waar Lamping op doelt.

Hoewel de expliciete weergave het bereik van een abstractor een rol speelt in de vertaling van λ-expressies in grafen waar Lambdascope mee werkt, stelt die rol de betekenis van haakjes in die grafen niet veilig: met het verdwijnen van een abstractor (32) kunnen we een haakje namelijk niet meer als het einde van het bereik van een abstractor zien.

We kunnen niet volhouden dat haakjes in grafen hetzelfde betekenen als het einde van het bereik van een abstractor. Want zelfs zonder abstractor hebben we de nieuwe grenzen, en de grenzen die aftakkingen aangeven nodig. De grenzen hebben een effect op de niveaus

die we gebruiken om namen van poorten van waaiers te onthouden.

Omdat we, wanneer we laten zien dat een algoritme werkt, dat wil zeggen, wanneer we de correctheid van dat algoritme bewijzen, knopen en grafen met expressies verbinden, geven we sowieso een betekenis aan grafen. We zien daarom geen noodzaak knopen in grafen nog meer betekenis te geven dan ze vanwege correctheid en binnen het algoritme zelf hebben.

We laten daarom zien dat we de regels van Lambdascope ook kunnen beschrijven zonder gebruik te maken van 'bereik'. Daarbij gaan we, net als Asperti en Guerrini, uit van het idee dat takken die naar de instanties van variabelen leiden via waaiers in de abstractor eindigen die bij de variabele hoort. Vanwege Lambdascope verbinden we niet de waaier of de applicator, maar juist de abstractor met de niveaus waarop we de namen van poorten onthouden.

De verschillende onderdelen van de beschrijving

De beschrijving van de regels van Lambdascope bestaat uit een aantal onderdelen. In ieder onderdeel bekijken we de regels in het algoritme vanuit een ander perspectief.

Omdat we moeten laten zien dat de stappen die we vanwege Lambdascope zetten niet van de reductie λ-expressies mogen afwijken, moeten we de manier waarop het algoritme grafen verandert vastleggen. We beschrijven daarom eerst een raamwerk waarin we die veranderingen vast kunnen leggen: in onderdeel 2 we kijken naar interactienetten.

Net als het algoritme van Asperti en Guerrini werkt Lambdascope met grafen waarin je de instantie van een variabele in de bijbehorende abstractor leest. Omdat we binding een rol laten spelen in het algoritme, geven we in onderdeel 3 de regels voor de interactie die de syntax van gewone λ-expressies weergeven verandert in de grafen die Lambdascope nodig heeft.

Zonder de stapel van niveaus kunnen we niet achterhalen welke λ-expressie een graaf weergeeft. De stapel vormt de context waarin we een graaf lezen die we met optimale stappen veranderen. In onderdeel 4 gaan we na op welke manier we namen van poorten kunnen onthouden.

Omdat de manier waarop we poorten van waaiers onthouden afhangt van de interactie tussen een waaier en een abstractor, bespreken we in onderdeel 4 ook de regels die de bewerkingen op de stapel bepalen. Maar met deze regels alleen lukt het niet de reductie in de λ-rekenkunde te volgen. In onderdeel 5 voegen we daarom extra regels voor interactie toe, en we laten ook zien dat we daarmee genoeg regels hebben.

Het feit dat een interactienet de reductie kan volgen zegt echter nog niet alles. Zouden we namelijk een expressie in een graaf lezen die we niet tegenkomen bij de reductie van

de expressie die we in de oorspronkelijke graaf lezen, dan werkt de interactie blijkbaar niet goed. In onderdeel 6, 'grafen lezen', leggen we daarom een afbeelding van grafen op expressies vast, een afbeelding waarmee we laten zien dat interactie doet wat we ervan verwachten.

In onderdeel 7 kijken we naar de betekenis die het algoritme in de praktijk van het rekenen met λ-expressies heeft. Naast het correcte en volledige bespreken we het feit dat we de stappen die we met Lambdascope zetten als optimaal mogen opvatten. We kijken ook naar de efficiëntie van het algoritme. Daarnaast geven we nog regels voor de interactie die grafen weer in bomen verandert.

2 Interactienetten

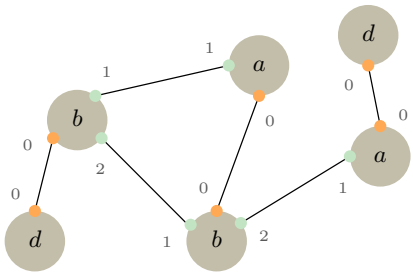
We hebben gezien dat we in een graaf met gebogen takken en waaiers het weglaten van een applicator en een daar aan voorafgaande abstractor, als een bewerking op kunnen vatten die overeenkomt met een optimale stap in de reductie van een λ-expressie.

Omdat waaiers echter niet in de syntaxboom van een expressie zitten, moeten we de boom veranderen in een graaf met waaiers. Daarbij moet er het nodige gebeuren, want we moeten de takken die bij een abstractor horen bij elkaar zoeken, en via een opeenvolging van waaiers met die abstractor verbinden.

Aangezien naast de bewerking die de we als optimale stap zien, ook het toevoegen van waaiers een graaf verandert, kijken we eerst op een abstracte manier naar verandering in grafen. Daarna leggen we het toevoegen van waaiers en de optimale stap en de interactie die deze met zich meebrengt vast.

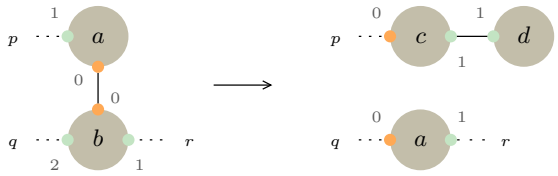
Interactie in een graaf

Een interactienet zoals Lafont [12] dat voorstelt, bestaat uit een graaf die vergezeld gaat van een verzameling regels die vastleggen hoe die graaf kan veranderen. Net als een gewone graaf heeft de graaf in een interactienet knopen, kanten en labels. Maar de graaf in een interactienet heeft ook poorten: de verbinding van een kant met een knoop. We geven poorten met natuurlijke getallen aan.



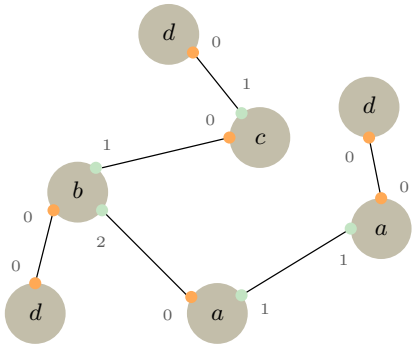
(36)

De 0-poort duidt de primaire poort van een knoop aan. Wanneer een knoop met een kant via twee 0-poorten aan een andere knoop vastzit, dan noemen we die knopen interactief: de interactie zet dan een stap door een regel te volgen die ze op die twee knopen toe kan passen. Bevat een interactienet de graaf (36) en de regel



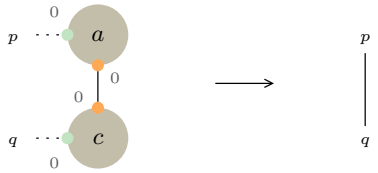
ab (37)

dan kan de interactie in de graaf de ab-regel volgen omdat de labels van de knopen in het linkerdeel van de regel overeenkomen met de labels van de knopen in de graaf. De knopen met label a en b verdwijnen uit de graaf. Het rechterdeel van de regel neemt hun plaats in:



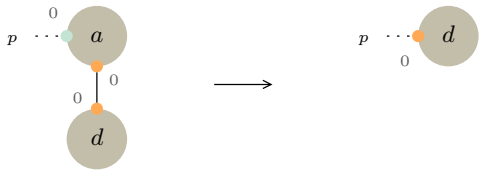
(38)

De interactie in de graaf is lokaal: wanneer interactie een regel volgt, verdwijnen twee interactieve knopen, de rest van de graaf blijft hetzelfde. Met de letters bij de uiteinden van de losse takken duidt een regel aan hoe de interactie de losse takken met de rest van de graaf moet verbinden.



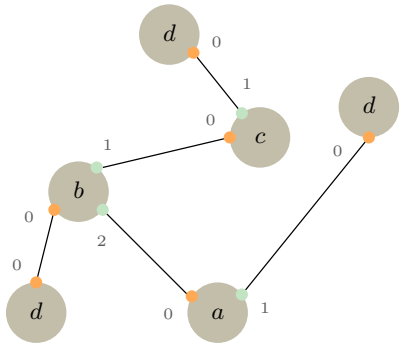
ac (39)

Wanneer interactie de ac-regel volgt, verbindt ze de poort waarmee knoop p aan de knoop met label a vastzit met de poort van de knoop die aan c vastzit. Omdat de knopen a en c niet interactief zijn, is de regel niet van toepassing op de graaf.



ad (40)

De interactie kan de ad-regel wel volgen:



(41)

Na de interactie zitten er nog steeds interactieve knopen in de graaf, maar als alleen de ab, ac, en ad-regel bij de graaf horen, volgt er geen interactie meer; we vatten de graaf als een normaalvorm op.

De verzameling regels die bij een interactienet hoort voldoet in ieder geval aan de volgende voorwaarden: bij iedere tak moet een letter staan, een letter in het linkerdeel moeten we in het rechterdeel terugvinden, en in beide delen mag een letter maar één tak staan. Verder mag het paar labels in het linkerdeel van een regel maar in één regel voorkomen.

De eigenschappen van een interactienet

Met een interactienet kun je de verandering van een graaf vastleggen. De interactie in een graaf die bij een interactienet hoort heeft gunstige eigenschappen.

Eigenschap: **interactie is lokaal** (42)

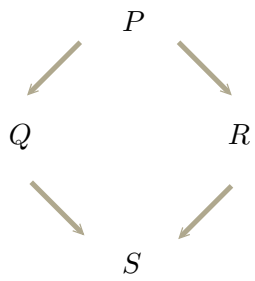
Met de interactie van twee interactieve knopen a en b verdwijnen alleen die knopen uit de graaf. De interactie vervangt a en b door een samenstelling die de plaats van deze knopen inneemt.

Als je naar een verandering kijkt, hoef je geen rekening te houden met de rest van de graaf; je hoeft die verandering niet op de rest van de graaf te betrekken. Dit maakt de definitie die

een verandering weergeeft eenvoudig. Bovendien maakt een lokale verandering parallelle stappen mogelijk.

Eigenschap: **interactie is sterk confluent** (43)

Naast twee interactieve knopen a en b bevat een graaf geen knoop c zodanig dat a en c of b en c ook interactief zijn. Vanwege het lokale beperkt de interactie van a en b de rest van de interactie niet. Als a en b op elkaar inwerken, verandert dit het interactieve van c en d niet: interactie is sterk confluent.



(44)

Als interactie de graaf P de ene keer in een graaf Q en de andere keer in een graaf R verandert, dan volgt vanwege de confluentie dat de interactie zowel in Q als in R een regel kan volgen, zodanig dat de interactie in beide gevallen in één en dezelfde graaf S eindigt.

Vanwege de sterke confluentie hoef je, wanneer je de verandering van grafen in de vorm van interactie beschrijft, geen rekening te houden met de volgorde van de stappen die de graaf veranderen. De manier waarop een graaf uiteindelijk verandert is niet afhankelijk van de volgorde van die stappen.

Eigenschap: **interactie verloopt parallel** (45)

Als a en b interactief zijn, en c en d ook, en de kanten $\langle a, c\rangle$, $\langle a, d\rangle$, $\langle c, d\rangle$ en $\langle b, d\rangle$ niet in de graaf zitten, dan kan de interactie vanwege het lokale naast de regel die van toepassing is op a en b ook de regel volgen die bij c en d hoort.

Tenzij de interactie een regel niet kan volgen vóórdat ze een andere regel heeft gevolgd, kan ze die stappen naast elkaar zetten.

Eigenschap: **interactie is asynchroon** (46)

Stappen die de interactie naast elkaar zet hoeft ze niet op hetzelfde moment te zetten.

Het asynchrone houdt in dat de interactie geen voorziening met zich mee hoeft te brengen die ervoor zorgt dat ze stappen die ze naast elkaar kan zetten ook op precies hetzelfde moment zet.

Interactienetten als structuur

Wanneer we naar een interactienet kijken zien we knopen en kanten, we zien ook de labels die bij de knopen horen. Omdat we onafhankelijk van de labels willen aangeven hoe een interactienet eruitziet, en niet voor ieder labeling een aparte definitie willen geven, leggen we een interactienet vast in de vorm van een structuur.

We vatten eerst een graaf als structuur op.

Definitie: **$\mathcal{L}$-graaf** (47)

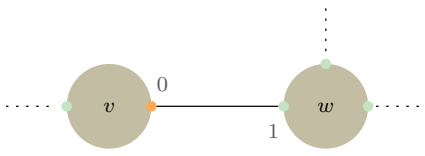
Met

> V een eindige verzameling, de knopen in de graaf, en
>
> $E \subseteq V \times V$ de kanten van de graaf, en
>
> $\ell : V \to \mathcal{L}$ een afbeelding die labels, elementen van een verzameling $\mathcal{L}$, aan de knopen van de graaf toekent,

is een $\mathcal{L}$-graaf een structuur $\langle V, E, \ell \rangle$ □

Een $\mathcal{L}$-structuur geeft de relatie tussen de labels weer. Als we een verzameling labels kiezen, dan hebben we met een $\mathcal{L}$-structuur ook een graaf; je kunt de labels als parameter van de structuur zien.

De grafen in een interactienet hebben poorten: een poort zit aan het uiteinde van een kant. Kanten zonder poorten zijn er niet, en in iedere poort kan er ten hoogste één kant beginnen of eindigen. Een kant ligt eenduidig vast wanneer we niet alleen zeggen welke knopen ze verbindt, maar ook via welke poorten ze dat doet.



$\{\langle v, 0\rangle, \langle w, 1\rangle\}$ (48)

De verzameling die uit $\langle v, 0\rangle$ en $\langle w, 1\rangle$ bestaat beschrijft de kant tussen v en w eenduidig.

Met een relatie

$$\alpha \subseteq \mathcal{L} \times \mathbb{N}$$

ligt vast hoeveel poorten een knoop die een bepaalt label draagt kan hebben. Omdat α niet noodzakelijk een afbeelding is, staan we toe dat het aantal poorten van een knoop met een bepaald label varieert. Naast $\mathcal{L}$ vatten we ook α als een parameter op; we leggen $\mathcal{L}$ en α nog niet vast.

Definitie: $\mathcal{L}\alpha$**-graaf** (49)

Met

V een eindige verzameling, de knopen in de graaf, en

$E \subseteq \{\{q, r\} \mid q, r \in V \times \mathbb{N}\}$ de kanten van de graaf, en

$\ell : V \to \mathcal{L}$ een afbeelding die labels, elementen van een verzameling $\mathcal{L}$, aan de knopen van de graaf toekent, en

$\alpha \subseteq \mathcal{L} \times \mathbb{N}$ en een relatie die vastlegt hoeveel poorten een knoop met een zeker label kan hebben,

is een $\mathcal{L}\alpha$-graaf een structuur $\langle V, E, \ell \rangle$ waarin

$$\wedge\ \forall_{v \in V} \exists_{\langle \ell(v), j \rangle \in \alpha} \forall_{i<j} (\exists!_{e \in E} (\langle v, i \rangle \in e))$$

$$\wedge\ \forall_{e_1, e_2 \in E} (e_1 \cap e_2 \neq \varnothing \to e_1 = e_2)$$

□

De eerste clausule beschrijft hoe de kanten eruitzien, de andere twee clausules drukken uit dat slechts één kant een knoop met een andere of met zichzelf verbindt.

Definitie: **equivalantie van** $\mathcal{L}\alpha$**-grafen** (50)

We beschouwen twee $\mathcal{L}\alpha$-grafen $\langle V_1, E_1, \ell_1 \rangle$ en $\langle V_2, E_2, \ell_2 \rangle$ als 'hetzelfde', dan en slechts dan als er een bijectieve afbeelding $f_{eq} : V_1 \to V_2$ bestaat zodanig dat:

$$\forall_{v_1, w_1 \in V_1} \forall_{v_2, w_2 \in V_2} \forall_{n, m \in \mathbb{N}} ($$

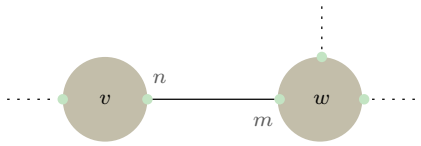
$$\ell_1(v_1) = \ell_2(f_{eq}(v_1)$$

$$\wedge\ \{\langle v_1, n \rangle, \langle w_1, m \rangle\} \in E_1 \to \{\langle f_{eq}(v_1), n \rangle, \langle f_{eq}(w_1), m \rangle\} \in E_2$$

$$\wedge\ \{\langle v_2, n \rangle, \langle w_2, m \rangle\} \in E_2 \to \{\langle f_{eq}^{-1}(v_2), n \rangle, \langle f_{eq}^{-1}(w_2), m \rangle\} \in E_1)$$

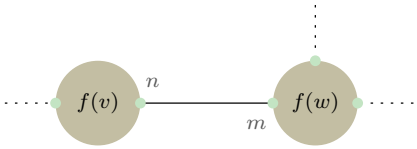
□

De tweede en derde clausule drukken uit dat er een afbeelding f_{eq} bestaat, zodanig dat voor iedere kant



$$\{\langle v, n\rangle, \langle w, m\rangle\} \tag{51}$$

in de ene $\mathcal{L}\alpha$-graaf,



$$\{\langle f_{eq}(v), n\rangle, \langle f_{eq}(w), m\rangle\} \tag{52}$$

in de andere graaf zit. Het spreekt niet vanzelf dat een kant die in de tweede graaf zit, ook deel uitmaakt van de eerste. Want knopen met hetzelfde label hebben niet noodzakelijk hetzelfde aantal poorten. De vierde clausule geeft aan dat een kant in de tweede graaf ook in de eerste zit.

Vanwege f_{eq} doen we afstand van de knopen van de graaf: we kunnen altijd een graaf vinden die andere knopen en kanten heeft maar die we toch als hetzelfde beschouwen; de knopen van $\mathcal{L}\alpha$-grafen zijn abstract.

De grafen die we in een interactieregel zien lijken veel op $\mathcal{L}\alpha$-grafen, met dat verschil dat we sommige knopen niet zien. Wat het vervangen van knopen betreft zouden we de kanten die in de onzichtbare knopen eindigen weg kunnen laten. Omdat we echter vast moeten leggen hoe de nieuwe knopen aan de rest van de graaf vastzitten, hebben we de kanten toch nodig.

Definitie: **partiële $\mathcal{L}\alpha$-graaf** (53)

Een partiële $\mathcal{L}\alpha$-graaf $\langle V_1, V_2, E, \ell\rangle$ is een $\mathcal{L}\alpha$-graaf $\langle V_2, E, \ell\rangle$ met $V_1 \subset V_2$ waarvan we de knopen die wel in V_2 maar niet in V_1 zitten niet zien. □

De verzameling V_1 beperkt de zichtbaarheid van knopen. Met het onzichtbare bedoelen we dat het niet uitmaakt welk label een dergelijke knoop heeft. Niet bij het vastleggen van de regel, en ook niet bij het zoeken naar een fragment in de graaf dat overeenkomt met het linkerdeel van een regel.

Definitie: **interactieregel** (55)

Met Π_1 de eerste component van de projectie van partiële $\mathcal{L}\alpha$-grafen g_1 en g_2 waarvoor

$$\forall_{v_1,v_2,v_3 \in \Pi_2(g_1)}(v_1 \neq v_2 \wedge v_2 \neq v_3 \rightarrow v_1 = v_3)) \text{ en}$$

$$\forall_{v_1,v_2 \in \Pi_2(g_1)}(\{\langle v_1, 0\rangle, \langle v_2, 0\rangle\} \in \Pi_3(g_1) \wedge v_1 \neq v_2)$$

en een bijectieve afbeelding f, gegeven door

$$\{\langle v_1, v_3\rangle \mid$$

$$(\{\langle v_1, n_1\rangle, \langle v_2, n_2\rangle\} \in \Pi_3(g_1) \wedge v_1 \notin \Pi_2(g_1)) \wedge v_2 \in \Pi_1(g_1))$$

$$\wedge\ (\{\langle v_3, n_3\rangle, \langle v_4, n_4\rangle\} \in \Pi_3(g_2) \wedge v_3 \notin \Pi_2(g_2)) \wedge v_4 \in \Pi_1(g_2))\}$$

is geordend paar $\langle g_1, g_2\rangle$ een interactieregel. De graaf g_1 is het linkerdeel van de regel, het deel dat interactie door g_2, het rechterdeel van de regel vervangt. □

De eerste clausule beperkt het aantal zichtbare knopen in de eerste partiële graaf tot twee, en de tweede clausule geeft aan dat een kant de primaire poorten van die knopen met elkaar moet verbinden. De derde clausule beschrijft de afbeelding die de niet zichtbare knopen van de ene partiële graaf aan die van de andere toevoegt. Met deze afbeelding geeft een regel aan hoe een interactiestap de nieuwe knopen met de rest van de graaf verbindt.

Het linkerdeel van twee regels in een verzameling interactieregels mag equivalent zijn, maar alleen dan als het rechterdeel dat ook is. Zou dit niet het geval zijn, dan legt een regel niet eenduidig vast hoe een interactiestap de knopen moet vervangen. Met andere woorden: de verzameling interactieregels moet deterministisch zijn.

Definitie: **determinisme** (56)

Een verzameling interactieregels R is deterministisch, dan en slechts dan als

$$\forall_{r_1,r_2 \in R}(\Pi_1(r_1) \equiv \Pi_1(r_2) \rightarrow \Pi_2(r_1) \equiv \Pi_2(r_2))$$

□

Op basis van de definitie van een $\mathcal{L}\alpha$-graaf, en de definitie van een interactieregel, leggen we vervolgens vast hoe een interactienet eruitziet.

Definitie: **interactienet** (57)

Een interactienet is een geordend paar $\langle g, R\rangle$ waarin

- g een $\mathcal{L}\alpha$-graaf, en

- R een deterministische verzameling interactieregels bij een $\mathcal{L}\alpha$-graaf is.

□

Een interactiestap vervangt twee interactieve knopen door het rechterdeel van de regel waarvan het linkerdeel met die knopen overeenkomt. De afbeelding f die deel uitmaakt van de interactieregel (55) bepaalt hoe het fragment in het rechterdeel van de regel in de graaf past.

Definitie: **interactiestap** (58)

Met N en de verzameling van interactienetten is een interactiestap een element van

$$S \subset N \times N$$

Een element $\langle\langle g_1, R\rangle, \langle g_2, R\rangle\rangle$ van S vind je als volgt.

- Als de verzameling regels R in een interactienet $\langle g_1, R\rangle$ een graaf bevat waarvoor geldt dat één of meerdere knopen van die graaf ook in g_1 zitten, begin dan met een graaf g_2 waarvoor

  $$g_2 \equiv g_1$$

  Stel in het andere geval g_2 gelijk aan de graaf in het interactienet. We kijken alleen nog naar g_2 en niet meer naar de oorspronkelijke graaf.

- Zoek een kant

  $$\{\langle v_1, 0\rangle, \langle v_2, 0\rangle\}$$

  in de graaf en een regel $r \in R$ zodanig dat de labels van v_1 en v_2 gelijk zijn aan de labels van de knopen in die regel. Als een dergelijke kant niet in de graaf zit, dan zit er geen element in de relatie S dat $\langle g_1, R\rangle$ als eerste component heeft; in $\langle g_1, R\rangle$ volgt er geen interactie.

  Haal in het andere geval de kant uit de graaf en ga verder.

- Verwijder de knopen v_1 en v_2 en voeg de kant $\Pi_2(r)$ aan de graaf toe.

- Neem een kant met v_1 en v_3 in de graaf. Met die kant komt een kant in het linkerdeel van de regel overeen: één van de knopen in die kant heeft hetzelfde label als v_1 terwijl de andere knoop onzichtbaar is. De afbeelding $\Pi_3(r)$ verbindt die knoop met een onzichtbare knoop v_5 in een kant met v_4 en v_5 in het rechterdeel van de regel.

  Haal de kanten met v_1 en v_4 weg, en voeg een kant met v_3 en v_4 toe.

- Voer de vierde stap ook uit voor alle knopen v_3 waar v_1 aan vastzit.

· Voer de vierde stap ook uit voor de kanten waar de knoop v_2 deel van uitmaakt.

□

De eerste clausule zorgt ervoor dat de knopen in de graaf verschillen van de knopen in de interactieregels. Dit maakt het mogelijk dat we het fragment waar de interactie betrekking op heeft door het rechterdeel van de regel kunnen vervangen. De derde clausule beschrijft die vervanging.

De vierde clausule beschrijft de verbinding van het rechterdeel van de interactieregel met de oorspronkelijke graaf. Omdat je zowel in het oorspronkelijke als het nieuwe fragment maar één kant losmaakt, komt er zowel bij v_3 als bij v_4 maar één poort vrij, en daardoor kun je de knopen maar op één manier met elkaar verbinden.

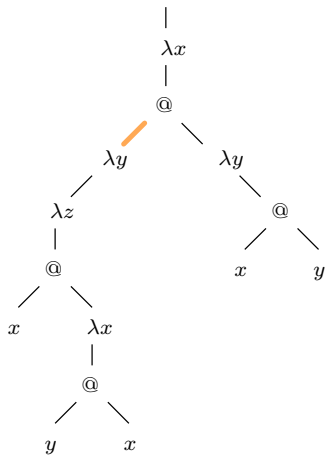
Definitie: **interactie** (59)

Met S de verzameling stappen (58) is interactie in een net een verzameling $S' \subset S$ waarin ieder element, een stap, deel uit maakt van een opeenvolging: na een stap $\langle a, b\rangle \in S$ kan iedere stap $\langle b, c\rangle \in S$ de volgende stap in die opeenvolging zijn. Als er geen c is zodanig dat $\langle b, c\rangle \in S$ dan is $\langle a, b\rangle$ de laatste stap in een opeenvolging. □

We geven een voorbeeld van interactie door een stap in de reductie van een expressie in de De Bruijn rekenkunde de vorm van een interactienet te geven.

De De Bruijn rekenkunde

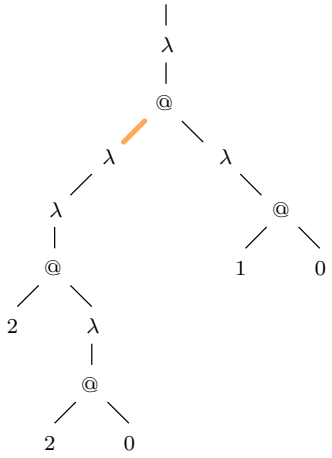
De expressies in de λ-rekenkunde hebben een nadeel: bij substitutie kan een abstractor in de abstractie van de applicatie een variabele in het argument binden die niet bij een abstractor in dat argument hoort.



(60)

Bij de reductie van de expressie die we in de boom lezen, vervangt het argument de variabele y in de abstractie. Daarmee bindt λx de variabele x in het argument. Dit terwijl de eerste λx in de boom dat vóór de reductie doet. De reductie verandert de binding alleen dan niet als de naam van de variabele die λx bindt vóór de substitutie verandert.

Bij een abstractor in een De Bruijn expressie [4] hoort geen naam, en ook de instanties van de variabele die bij de binder horen hebben geen naam. We vatten ze als indices op: met een natuurlijk getal dat de afstand en de abstractor weergeeft duidt een instantie van een variabele haar binder aan.



(61)

Als we de natuurlijke getallen als het aantal abstractors tussen de index en de abstractor die haar bindt interpreteren, en iedere keer wanneer we door een abstractor lopen de naam van een verse variabele op een stapel leggen, dan volgt dat we in de abstractie van de eerste applicatie

$$\lambda x.\lambda y.w(\lambda z.xz)$$

en het argument

$$\lambda x.wx$$

lezen.

Hoewel we in de De Bruijn rekenkunde bij reductie van een expressie geen nieuwe namen hoeven te verzinnen, moeten we er wel voor zorgen dat de indices na die reductie naar dezelfde abstractors wijzen als ervoor.

De index 1 in het argument van de buitenste, eerste applicatie (61) bevat een index die verder wijst dan de eerste applicator, de applicator die bij reductie verdwijnt. Wanneer het argument de plaats van de variabele inneemt, zou de index 1 naar een andere abstractor wijzen als vóór de reductie.

Alleen als we de index door een nieuwe verwijzing naar de binder waar de index naar wijst

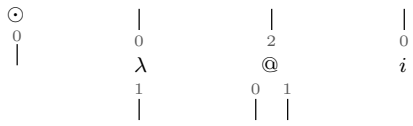
vervangen, verandert de binding in het argument niet. Daarbij moeten we er rekening mee houden dat er vanwege de reductie een abstractor verdwijnt. Omdat er uiteindelijk twee abstractors tussen de index en haar binder zitten, geven we de knoop 2 als label.

Maar ook de indices in de abstractie blijven niet altijd hetzelfde. Als een index verder wijst dan de abstractor die bij de reductie verdwijnt, moeten we die index met één verlagen. Na de reductie zit er namelijk één abstractor minder tussen de index en de abstractor waar deze naar wijst.

De expressies in de De Bruijn rekenkunde als interactienetten

We geven reductie van een expressie in de De Bruijn rekenkunde de vorm van interactie in een graaf waarin er één applicator en één abstractor met de primaire poorten elkaar vastzitten. De interactie in een dergelijke graaf kan dus maar met één stap in de reductie van die expressie overeenkomen.

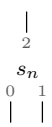
Wanneer we een boom lezen die een De Bruijn expressie weergeeft komen we daarin de volgende knopen tegen.



(62)

Als eerste is er de wortel, daarna komen we verschillende applicators en abstractors tegen, en aan het einde van iedere tak komen we tenslotte bij een index. De wortel en de index hebben beide één poort, de abstractor twee, en de applicator drie.

Wanneer een applicator op een abstractor inwerkt, verdwijnt zowel de applicator als de abstractor. Omdat de interactie het argument naar de binders moet brengen, introduceren we een nieuwe knoop, een knoop die s als label heeft. Deze knoop brengt de 1-tak van de applicator naar de indices die bij de abstractor horen die bij de reductie verdwijnt.

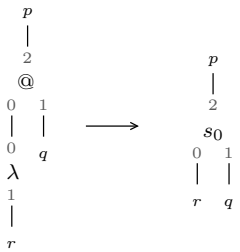


(63)

De s-knoop telt ook abstractors. Ze onthoudt een aantal n doordat de interactie haar label

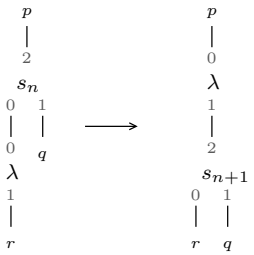
uitbreidt. We hebben daarom niet genoeg aan één s-knoop: voor iedere $n \in \mathbb{N}$ is er een knoop die s_n als uitgebreid label heeft. Omdat de knoop zich op weg naar de indices tussen de andere knopen in de boom moet begeven, heeft ze in ieder geval twee poorten nodig. De 1-tak van de applicator zit aan een derde poort vast.

Wanneer een applicator een 0-tak heeft, en die tak met een abstractor begint, lezen we een reduceerbare expressie. Bij de reductie verdwijnen de applicator en de abstractor, en het argument, de 1-tak van de applicator, moet de plaats innemen van de instanties van de variabele die de abstractor bindt.



(64)

De s-knoop brengt het argument van de applicatie naar de indices. Daarbij springt ze over de knopen in de tak. Wanneer de s-knoop in de graaf verschijnt, heeft ze s_0 als uitgebreid label, en bij iedere abstractor die de knoop tegenkomt, telt de interactie één bij de waarde van de uitbreiding in het label op.

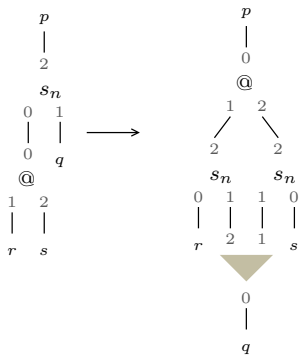


(65)

Wanneer de s-knoop bij een index komt, kan de interactie op basis van de uitbreiding van het label bepalen of een index verder wijst dan de abstractor die met de reductie uit de boom verdwijnt, en dus of ze de waarde van een index in de abstractie met één moet verlagen of niet.

Omdat er in beide takken van een applicator tenminste één index zit, gaat er, wanneer een s-knoop een applicator tegenkomt ook in beide takken een s-knoop verder. Als er niet

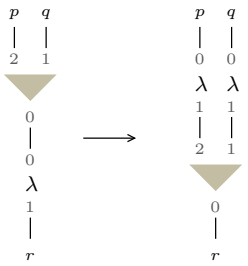
evenveel abstractors in de takken zitten, dan blijft het uitgebreide label van de s-knoop in die takken niet hetzelfde.



(66)

Aangezien beide s-knopen het argument naar het einde van de takken moeten brengen, voegt de interactie een waaier toe. Op die manier volgt in beide 1-takken nog steeds het argument. De interactie gebruikt de waaier niet op dezelfde manier als de interactie die optimale stappen zet. Bij de interactie in een De Bruijn boom gaat het niet om het op één plaats weergeven van de instanties van een variabele, maar om het op één plaats weergeven van het argument van een applicatie.

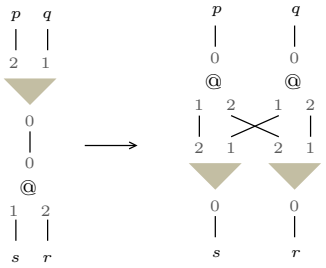
De waaier zorgt ervoor dat we in twee takken hetzelfde lezen. Omdat er uiteindelijk geen waaiers in de boom mogen blijven zitten, vervangt de interactie een waaier waarvan de 0-tak met een abstractor heeft door twee abstractors, één in iedere tak die naar de waaier leidt. De waaier gaat verder in de richting van de bladeren van de boom.



(67)

Als een waaier een abstractor tegenkomt, moet er net als bij de interactie van een waaier en een abstractor in iedere tak die naar de waaier loopt een applicator terechtkomen. Omdat

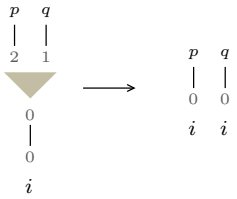
één applicator zelf al twee takken heeft, kan dit alleen als de interactie de 1-takken en de 2-takken van de applicators via twee waaiers verbindt.



(68)

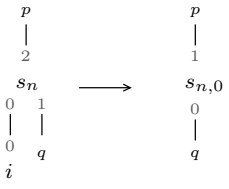
Wanneer je door de graaf loopt, kom je na interactie van de waaier en de applicator tussen p en r en tussen q en s dezelfde knopen tegen als vóór de interactie, zij het niet in dezelfde volgorde. Omdat we de waaier geen deel uitmaakt van de term, maakt de verandering van de volgorde geen verschil voor de term die je leest.

Aan het einde van een tak komt een waaier een index tegen. Omdat er in de boom uiteindelijk geen waaiers mogen blijven zitten, zorgt de interactie ervoor dat we in beide takken, zonder door een waaier te lopen, dezelfde index lezen. De waaier verdwijnt uit de boom.



(69)

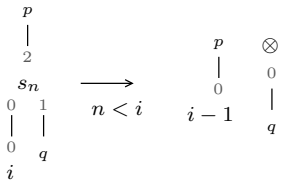
Oorspronkelijk (66) zit de waaier tussen de s-knoop en de index in een tak. Met het verdwijnen van de waaier kan de s-knoop eindelijk op een index inwerken. De uitbreiding in het label van de s-knoop is gelijk aan het aantal abstractors tussen die knoop en de abstractor die verdwijnt. Alleen als dit aantal gelijk is aan het label van de index vervangt de interactie de index door het argument.



(70)

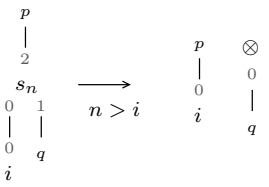
Als de interactie de index vervangt, gaat het in het argument een s-knoop verder. De interactie verandert het aantal poorten van de s-knoop, en ze breidt het label verder uit: naast het aantal abstractors tussen het argument en de abstractor die verdwijnt, onthoudt de s-knoop ook hoeveel abstractors ze in het argument tegenkomt vóór dat ze een index in het argument bereikt.

Als het label van de index niet gelijk is aan de uitbreiding van het label van de s-knoop, dan hoeft de interactie het argument niet op de plaats van de index neer te zetten. Wijst de index verder dan de abstractor die verdwijnt, dan past de interactie deze aan: de abstractor waar de index naar wijst ligt één abstractor minder ver weg.



(71)

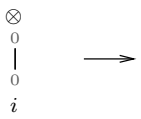
Wijst de index niet verder dan de abstractor die verdwijnt, dan verandert het verdwijnen van de abstractor niets aan de abstractor die de index aanduidt.



(72)

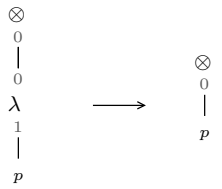
Omdat de interactie het argument niet meer nodig heeft, kan het argument, net als de s-knoop, uit de boom verdwijnen. Omdat er meerdere knopen in het argument kunnen zitten, voegt de interactie een knoop toe, een wisser, die alle knopen die volgen uit de graaf haalt.

De wisser moet iedere abstractor, applicator, en index in het argument verwijderen. Omdat een s-knoop en een waaier vanzelf uit de boom verdwijnen, hoeven we geen regel voor de interactie van een wisser en een dergelijke knoop toe te voegen. Als een wisser op een index inwerkt zien we geen van beide in de graaf terug.



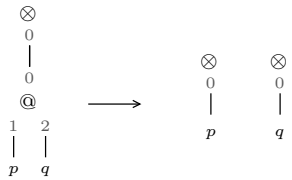
(73)

Werkt de wisser op een abstractor in, dan verdwijnt die uit de graaf. De wisser gaat in de tak van de abstractor verder.



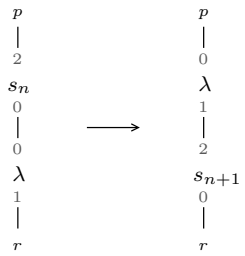
(74)

Omdat een applicator twee takken heeft, gaat er in iedere tak een wisser verder.



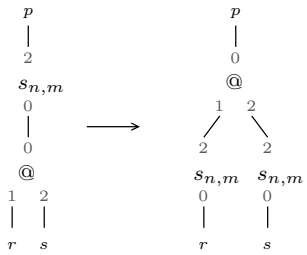
(75)

We kijken tenslotte naar de interactie die de indices in het argument bijwerkt. Niet anders dan in de abstractie gaat een s-knoop in de tak van een abstractor verder.



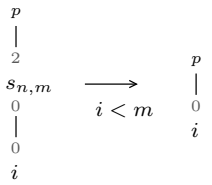
(76)

Maar anders dan in de abstractie hoeft een s-knoop in het argument niets naar de indices te brengen. Wanneer een s-knoop een applicator tegenkomt, heeft de interactie daarom geen waaier nodig.



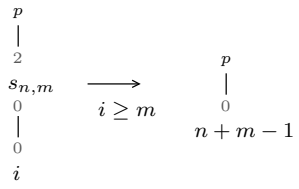
(77)

De s-knoop bereikt uiteindelijk een index in het argument. Als het label van die index kleiner of gelijk is aan de tweede uitbreiding van het label van de s-knoop, dan wijst de index naar een abstractor in het argument. De interactie hoeft in dat geval de index niet aan te passen.



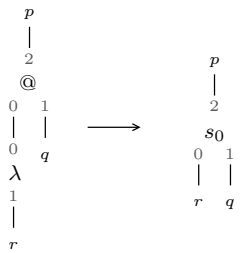
(78)

Als een index in het argument een abstractor aanwijst die aan het argument voorafgaat, moet de interactie één minder dan de abstractors die ze in de abstractie tegenkomt bij de waarde van die index optellen.

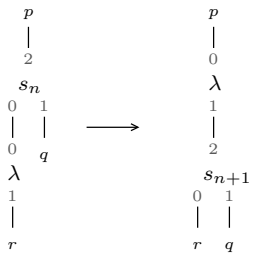


(79)

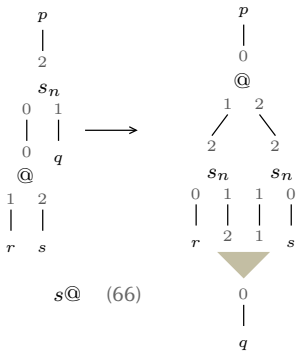
Aan deze regels hebben we genoeg wanneer we een reductie in de De Bruijn rekenkunde de vorm willen geven van een interactienet. We zetten alle interactieregels nog even op een rij.



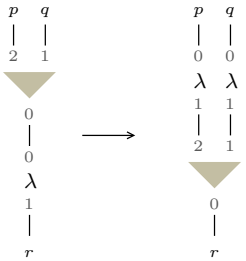
β (64)



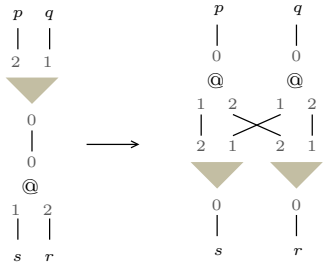
$s\lambda$ (65)



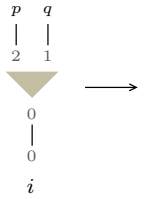
$s@$ (66)



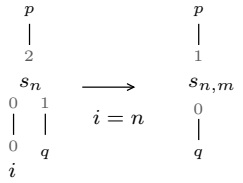
$\triangle\lambda$ (67)



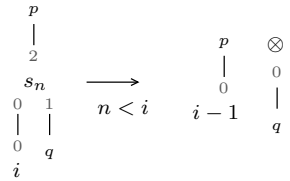
$\triangle@$ (68)



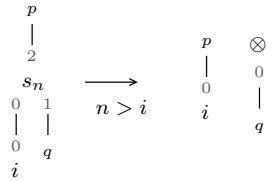
$\triangle i$ (69)



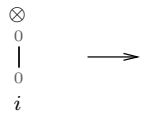
si (70)



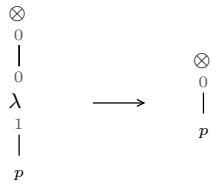
si (71)



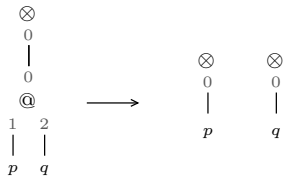
si (72)



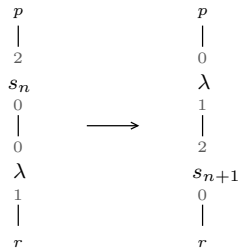
$\otimes i$ (73)



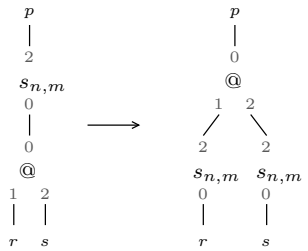
$\otimes\lambda$ (74)



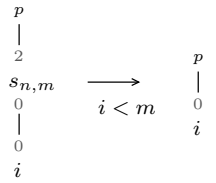
$\otimes @$ (75)



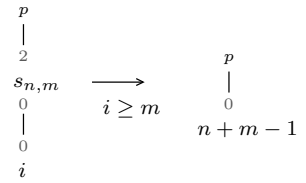
$s\lambda$ (76)



$s@$ (77)



si (78)



si (79)

Hoe een b-interactienet eruitziet

Met de letters bij de kanten in de weergaven (p.53, 54) van de partiële grafen duiden we onzichtbare knopen aan. We leggen de afbeelding die bij een interactieregel hoort vast door af te spreken dat die afbeelding knopen die dezelfde naam hebben aan elkaar toevoegt.

Als een weergave van een interactieregel een index of een s-knoop bevat, dan legt deze niet één interactieregel maar een hele verzameling vast. We zien de weergaven daarom ook als schema's. Voor ieder natuurlijk getal of ieder paar natuurlijke getallen leidt een schema tot een aparte regel.

De structuren die we hebben besproken kunnen we nog niet als een graaf, een verzameling interactieregels of een interactienet opvatten: de knopen, maar ook de afbeelding $\mathcal{L}$ die labels aan knopen toevoegt, en de relatie α die aangeeft hoeveel poorten een knoop kan hebben ontbreken nog. De structuren zijn leeg, abstract.

Wanneer we de structuren op basis van de schema's invullen krijgen we echte regels, een echte graaf en een interactienet waarin we kunnen zien hoe de interactie een reductie uitvoert. Omdat de expressie die we in de graaf lezen een expressie in de De Bruijn rekenkunde is, spreken we over een b-interactienet.

Definitie: **b-interactienet** (80)

Met

- $\mathcal{L}$ de verzameling labels

$$\{\odot, @, \lambda, i, s_n, s_{n,m}, \triangle, \otimes \mid i \geq 0, n \geq 0, m \geq 0\}$$

- $\alpha \subset \mathcal{L} \times \mathbb{N}$ de relatie

$$\{\langle \odot, 1\rangle, \langle @, 3\rangle, \langle \lambda, 2\rangle, \langle i, 1\rangle, \langle s_n, 3\rangle, \langle s_{n,m}, 2\rangle, \langle \triangle, 3\rangle, \langle \otimes, 1\rangle \mid i \geq 0, m \geq 0, n \geq 0\}$$

- g een $\mathcal{L}\alpha$-graaf, en

- R de verzameling interactieregels 64 tot en met 79,

is het geordende paar $\langle g, R\rangle$ een b-interactienet. □

Wanneer de structuur (57) invullen, we moeten laten zien dat we dat op een geschikte manier doen. Voor wat betreft de keuze van de graaf en de interactieregels zijn we niet gebonden. Maar we moeten er wel op letten dat de letters bij de kanten in het linkerdeel en het rechterdeel van een regel een bijectieve afbeelding vastleggen. Als dit het geval is, spreken we van een robuuste definitie.

Stelling: **robuustheid** (81)

De definitie van b-interactienet is robuust.

Bewijs: voor iedere $r \in R$ is $\Pi_3(r)$ een bijectie, en R is deterministisch. □

Definitie: **b-interactie** (81)

Neem een syntaxboom in de b-rekenkunde. Voorzie de knopen van poorten, zodanig dat alle knopen met de 0-poort naar de wortel van de boom wijzen. Kies een applicator die een 1-tak heeft die met een applicator begint. Verander de knopen van die applicator, zodanig dat deze met de 2-poort naar de wortel wijst, en de abstractor in de 0-tak zit.

Als we de boom als een $\mathcal{L}\alpha$-graaf g_1 opvatten, dan is b-interactie in het net $\langle g_1, R\rangle$ de opeenvolging van stappen die met een stap

$$\langle\langle g_1, R\rangle, \langle g_2, R\rangle\rangle$$

begint. □

Een voorbeeld van b-interactie

Bij wijze van voorbeeld kijken we naar de interactie in de graaf waarin we in beide takken van een applicator de Church-weergave van het natuurlijk getal 2 lezen. De weergave van 2 in de 1-tak van de applicator duiden we in het begin van de interactie met $\underline{2}$ aan.

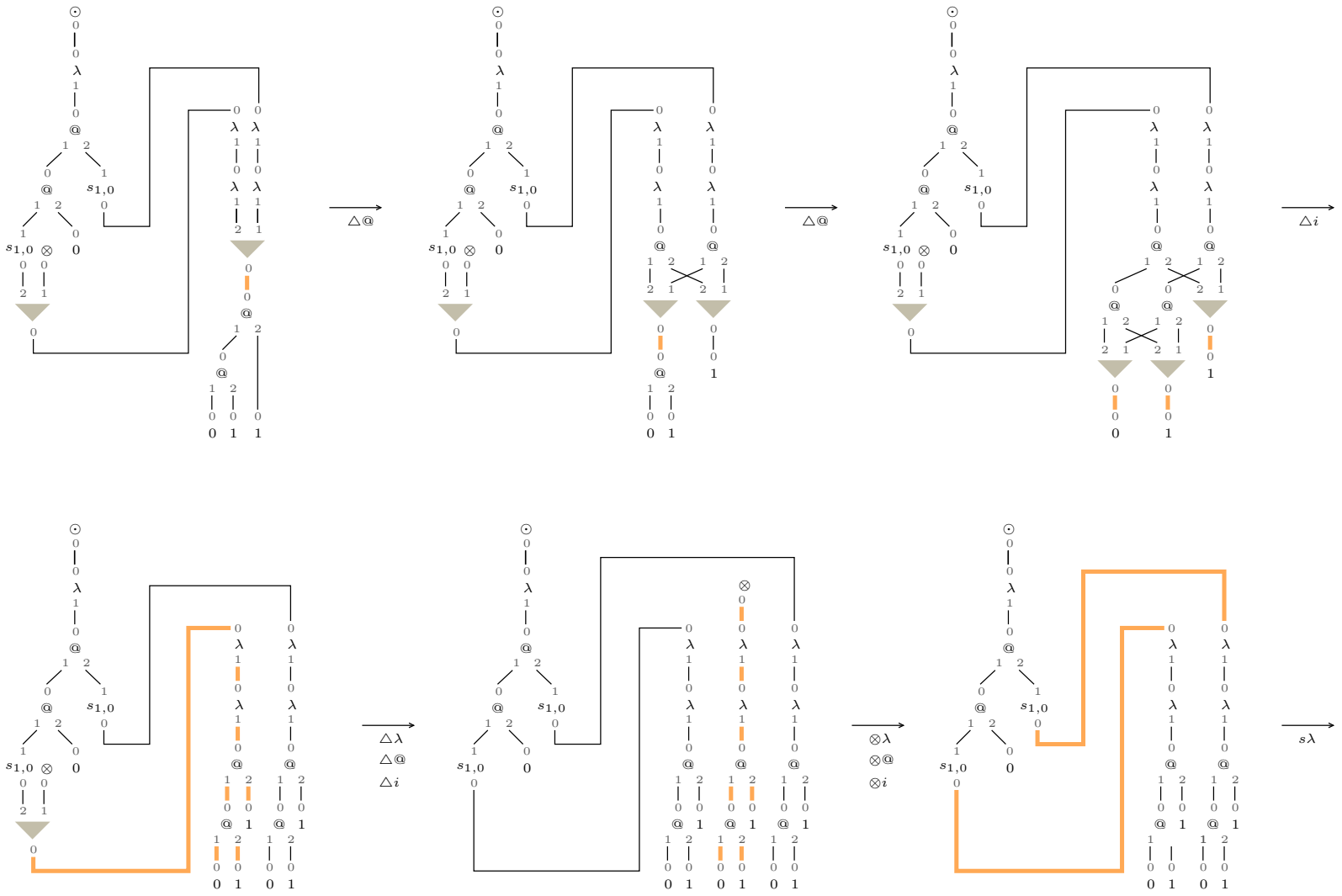
Wanneer we een reductie willen uitvoegen die overeenkomt met een berekening in de gewone rekenkunde, moeten we naast de natuurlijke getallen zelf, ook de bewerkingen daarop de vorm geven van een λ-expressie. Dit laten we achterwege; het voorbeeld heeft alleen betrekking op de weergave van de natuurlijke getallen.

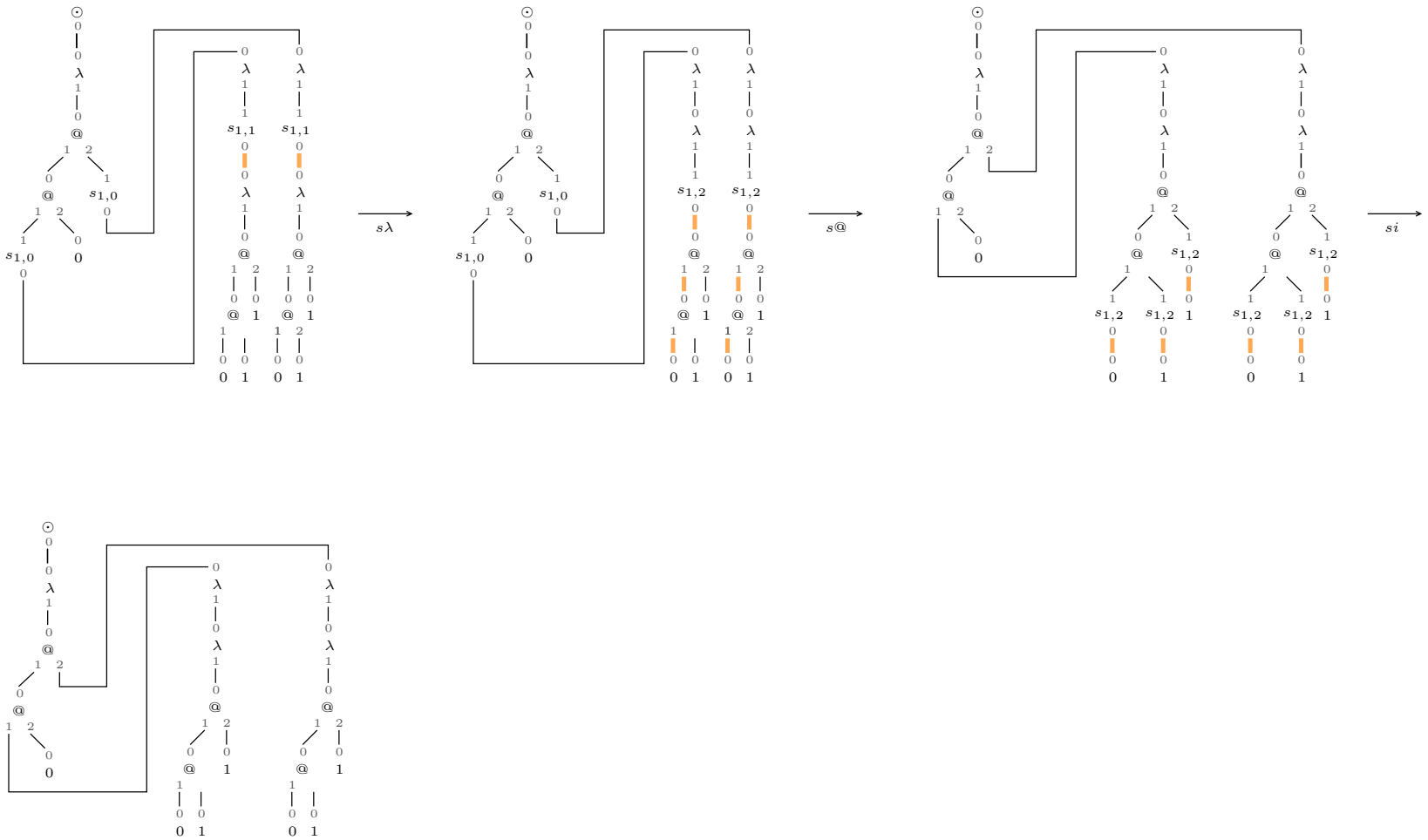
Omdat de applicator en de abstractor in de eerste graaf (p.57) met hun 0-poorten aan elkaar vastzitten, verdwijnen beide knopen en verschijnt er een s-knoop in de graaf. De interactie met de eerstvolgende abstractor geeft de s-knoop s_1 als label, want er zit in ieder geval één abstractor tussen de abstractor die verdwijnt en de indices in de takken.

Na de interactie van de s-knoop en de applicator gaat er in beide takken van de applicator een s-knoop verder. Bij die interactie verandert het uitgebreide label niet. De interactie heeft een waaier nodig om ervoor te zorgen dat we ook na de interactie in de 2-tak van beide s-knopen het argument lezen.

Bij de volgende interactie werkt de s-knoop op de index met label 0 in. Omdat die index een abstractor aanduidt die volgt op de abstractor die verdwijnt, verandert de interactie de waarde van de index niet. De s-knoop verdwijnt uit de graaf.

β
sλ
s@
s@
si
si
△λ
△λ


△@
△@
△i
△λ
△@
△i
⊗λ
⊗@
⊗i
sλ


$s\lambda$
$s@$
si

De overgebleven s-knopen werken daarna in op de indices die 1 als label hebben. Omdat een dergelijke index de abstractor aanduidt die verdwijnt, vervangt de s-knoop de index door de tak waarin we het argument lezen. De s-knoop die de tak in gaat zorgt ervoor dat de indices daar uiteindelijk weer dezelfde abstractor aanduiden.

Omdat de waaiers (p.58) de interactie van s-knopen en de indices in de weg zitten, maken ze plaats door op de applicators en de abstractors in het argument in te werken: we krijgen daardoor twee takken waarin we zonder een waaier tegen te komen het argument lezen.

Wanneer de nieuwe s-knoop in het argument (p.59) een applicator tegenkomt, gaat ze gewoon verder. De knoop onthoudt het aantal abstractors dat ze heeft geteld. Daarnaast houdt ze bij hoeveel abstractors er in het argument aan de index voorafgaan: bij iedere interactie met een abstractor telt de interactie één bij dit aantal op.

Zodra een s-knoop aan een index vastzit, kan ze de waarde in het label op de juiste manier aanpassen, zodanig dat de index de abstractor aanduidt die ze vóór de interactie van de applicator en de abstractor aanduidt.

Asynchroon en parallel

De interactie verloopt parallel als de interactie stappen naast elkaar kan zetten. Dit geldt in ieder geval voor de interactie (p.59) in takken van de graaf waarin we het argument van de applicatie lezen: de knopen in die takken zitten niet aan elkaar vast en de interactie is lokaal.

Het asynchrone houdt in dat de interactie niet op hetzelfde moment hoeft te beginnen aan stappen die ze naast elkaar kan zetten. Zo mag de interactie in het voorbeeld in de ene tak waarin we het argument lezen aan een stap zetten terwijl een stap in de andere tak nog niet klaar is.

Het parallelle en het asynchrone beperken zich niet tot verschillende takken. Hoewel de interactie in het voorbeeld eerst het hele argument kopieert, en daarna de s-knopen over de knopen in de instanties laat springen, zou een s-knoop ook onmiddellijk op een waaier kunnen volgen.

Het lezen van een interactienet

Wanneer we de interactie in een graaf bespreken, kijken we op een bepaalde manier tegen oriëntatie van de knopen aan. We zeggen bijvoorbeeld dat na een bepaalde interactie in de graaf de 0-tak van een applicator met de een abstractor begint, en dat die abstractor met haar 0-poort naar de applicator wijst. We spreken het volgende af over de manier waarop we het nodige over knopen, kanten en takken beweren.

Definitie: **opeenvolging van kanten** (85)

In een graaf $\langle V, E, \mathcal{L} \rangle$ is met $e_0, ..., e_n \in E$

$$e_0 \cdot ... \cdot e_n$$

een opeenvolging van kanten dan en slechts dan als

$$\forall_{0 \leq i < n}(\Pi_2(e_i) = \Pi_2(e_{i+1}))$$

□

De definitie van een opeenvolging van kanten staat los van het lezen van een graaf, want alleen de definitie van de graaf bepaalt de opeenvolging; we hoeven niet door de graaf te lopen.

Wanneer we het over een graaf hebben, en we de oriëntatie van de ene knoop ten opzichte van de andere bespreken, en bijvoorbeeld zeggen dat een knoop met haar 0-poort naar de wortel wijst, dan bedoelen we het volgende.

Definitie: **wijzen** (86)

Als een knoop v_1 met haar n-poort naar de een knoop v_2 wijst, dan betekent dit dat er een opeenvolging van kanten is die de n-poort van v_1 met een poort van v_2 verbindt. □

Wanneer we een graaf lezen, lopen we in een bepaalde richting door de knopen.

Definitie: **richting** (87)

Wanneer we bij het lezen van een graaf na de kant

$$\{\langle v_1, n_1 \rangle, \langle v_2, n_2 \rangle\}$$

de kant

$$\{\langle v_2, n_3 \rangle, \langle v_3, n_4 \rangle\}$$

volgen, dan lezen we de knoop v_2 in de richting

$$(n_2, n_3)$$

We gaan v_2 via de n_2-poort binnen, en we verlaten v_2 via de n_3-poort. Met

$$v_{(n_1, n_2)}$$

duiden we aan dat we de knoop v in de richting

$$(n_1, n_2)$$

lezen. □

Definitie: **tak** (88)

De n-tak van een knoop v is de opeenvolging van kanten die we bij het lezen van een graaf volgen wanneer we v via de n-poort verlaten. □

Wat hetzelfde blijft en wat niet

De interactie in het voorbeeld moet overeenkomen met een stap in de reductie van een expressie met De Bruijn indices. Dit houdt onder andere in dat iedere index in de graaf na de interactie dezelfde abstractor aan moet wijzen als vóór de interactie. Wanneer we de b-interactieregels inspecteren, merken we dat dit niet onmiddellijk volgt.

Omdat de interactie in de graaf verschillende stappen nodig heeft om ervoor te zorgen dat de indices op de juiste manier veranderen, kun je, door slechts naar één stap te kijken, niet laten zien dat de interactie haar werk op een correcte manier doet.

Wat b-interactie betreft komt het erop neer dat de uitbreiding in het label van een s-knoop met drie poorten gelijk moet zijn aan het aantal abstractors tussen de abstractor die verdwijnt en de s-knoop. Alleen als dit bij de interactie van s-knoop en een index het geval is, krijgt die index het juiste label.

Omdat iedere stap aan de voorwaarden moet voldoen die we aan de knopen in de graaf stellen, bijvoorbeeld de bewering over de uitbreiding in het label van de s-knoop, vormen die voorwaarden een invariante relatie: een inspectie van de regels voor de interactie moet uitwijzen of het volgen van een regel in een graaf die de relatie waar maakt, tot een graaf leidt die de relatie nog steeds waar maakt.

Naast het invariante moet de interactie de graaf op een bepaalde manier veranderen. Als de s-knopen zich bijvoorbeeld niet in de richting van de indices bewegen, dan doet de interactie niet wat we ervan verwachten. Naast een invariante relatie leggen we daarom ook een variante functie vast, een functie waarvan we laten zien dat de waarde bij iedere stap daalt.

Samen met de invariant impliceert het dalen van de variante functie een eigenschap van de graaf in een interactienet. Neem in het geval van b-interactie de som van de afstanden van de s-knopen tot de indices als variante functie. Omdat de functie daalt en de waarde nul krijgt, zit er uiteindelijk geen enkele s-knoop meer in de graaf. Houdt de interactie zich aan de invariant, dan duidt iedere index in de normaalvorm de juiste abstractor aan.

Optimale stappen en de introductie van de waaier

Wanneer b-interactie een waaier toevoegt dan doet ze dat omdat we het argument van een applicatie in plaats van in één, in twee takken moeten lezen. De introductie van een waaier volgt na de interactie van een applicator, namelijk wanneer de s-knoop die bij die interactie in de graaf verschijnt in twee takken verder gaat.

Omdat de introductie van de waaier op de interactie van een applicator en een abstractor volgt in plaats van eraan vooraf te gaan, kan de interactie in geen enkel geval de rol van een optimale stap vervullen.

Aangezien een graaf waarin de interactie optimale stappen kan zetten, vóór die stappen de nodige waaiers bevat, en we in een syntaxboom helemaal geen waaiers tegenkomen, moeten we nagaan hoe we waaiers aan dergelijke bomen toe zouden willen voegen. In onderdeel 3 leggen we, ook weer in de vorm van een interactienet, de verandering van bomen in een grafen met waaiers en gebogen takken vast.

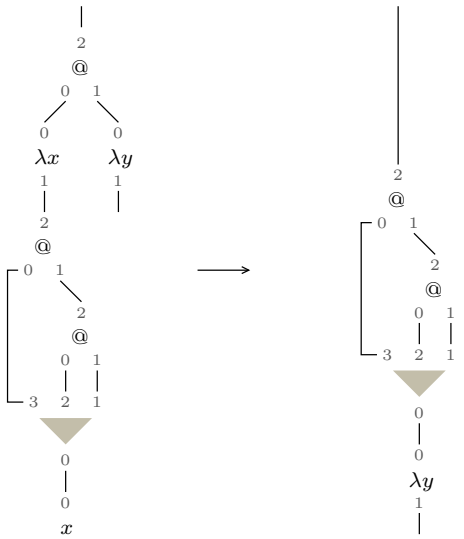
3 Naamloze en enkelvoudige binding

Asperti en Guerrini beschrijven een algoritme dat λ-expressies reduceert. Het alternatief dat we in onderdeel 4 en 5 voorstellen heeft ook de vorm van een interactienet, en werkt, net als het oorspronkelijke algoritme, met waaiers en enkelvoudige binding. Als waaiers of andere knopen een optimale stap in de weg zitten, maakt het nieuwe algoritme echter op een andere manier ruimte.

Omdat enkelvoudige lokale binding de basis van het algoritme is, leggen we eerst een algoritme vast dat de syntaxboom van een λ-expressie in een graaf met dergelijke binding verandert. Vanwege het enkelvoudige spreken we over e-interactie. We gaan na welke knopen we nodig hebben, en leiden de nodige regels voor de interactie af.

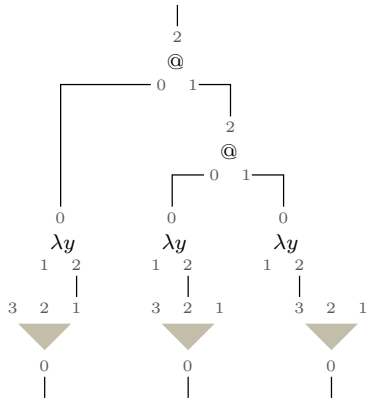
Waaiers en enkelvoudige binding

Een cascade van waaiers maakt binding enkelvoudig. Maar in plaats van een cascade zou je ook kunnen denken aan één waaier die tegelijk alle takken bindt. Want ook een waaier die meer dan twee takken samenbrengt kan een variabele op één plaats weergeven.



(89)

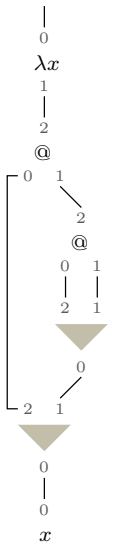
Na de interactie lezen we een reduceerbare expressie. Maar er zit een waaier tussen de applicator en de abstractor, en daarom kan de optimale stap alleen met het verdwijnen van die waaier volgen. Aangezien we de waaier niet lezen, zou ze ruimte kunnen maken door in de 0-takken van de applicators te springen.



(90)

Maar we merken dat het niet lukt alle poorten van de waaiers en de andere knopen in de graaf die nog niet aan een andere knoop vastzitten met elkaar te verbinden: de abstractors hebben samen nog drie vrije poorten, iedere waaier twee, samen zes.

Omdat alle takken uiteindelijk in dezelfde variabele eindigen, maakt het geen verschil of één waaier met vier poorten (89) de takken samenvoegt, of dat een opeenvolging van twee waaiers met ieder drie poorten dat doet.

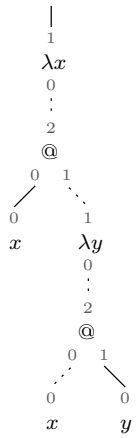


(91)

Zelfs als er nog meer takken samen moeten komen heeft e-interactie genoeg aan waaiers met drie poorten: voor iedere extra tak voegt ze één waaier aan de cascade toe.

Lokale binding

Een abstractor in de syntaxboom van een λ-expressie bindt de instanties van de variabele niet enkelvoudig: ze bindt verschillende bladeren die zich op verschillende plaatsen in de graaf bevinden. Omdat een blad niet samenvalt met haar binder, de abstractor, bindt die de instanties van een variabele ook niet lokaal.



(92)

Wanneer de interactie de takken die in bladeren eindigen die bij dezelfde binder horen via een cascade bij elkaar brengt, maakt ze de binding in de graaf enkelvoudig. Als de interactie er bovendien voor zorgt dat de cascade in de binder eindigt, maakt ze de binding in de graaf ook lokaal.

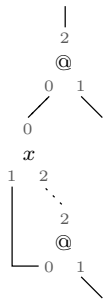
Als interactie een blad in een boom in een knoop verandert die hetzelfde label, maar twee extra poorten heeft, dan kan die knoop zich, in tegenstelling tot een blad, tussen de andere knopen begeven. Omdat de nieuwe knoop het uiteinde van de tak vasthoudt, buigt de tak vanzelf in de richting van de binders.

Zodra een variabele zich tussen de knopen in de graaf begeeft, en daarmee de tak waarin ze zit buigt, heeft de abstractor die de variabele bindt een gebogen tak. Wanneer we het over een gebogen tak van een abstractor hebben bedoelen we een tak die in de abstractor begint, en eindigt daar waar we een variabele lezen.

Wanneer we bij het lezen van de graaf door een knoop met een variabele als label lopen, mogen we alleen aan het einde van de tak een variabele lezen. Zouden we de variabele tussen de andere knopen in de graaf lezen, dan lezen we niet langer hetzelfde als in een boom.

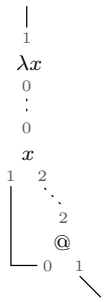
De interactie moet de takken die ze buigt uiteindelijk via een cascade van waaiers in één knoop samen laten komen. Als die knoop zich niet in de tak zou bevinden waarin de variabele zit, moet de variabele niet alleen in de tak waarin ze zit, maar ook in aftakkingen

naar die knoop zoeken.



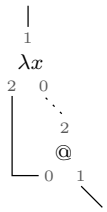
(93)

Als de interactie in de graaf de variabele echter niet naar een knoop ergens in de takken brengt, maar het uiteinde van een tak waarin we die variabele lezen naar haar binder buigt, komt de variabele, iedere keer wanneer ze over een knoop in de tak springt, dichter bij die binder. Tenzij we met een vrije variabele te maken hebben natuurlijk.



(94)

De variabele heeft haar binder gevonden zodra ze aan de 1-poort van een abstractor vastzit die de naam van de variabele in haar uitgebreide label heeft. Met nog één extra interactie kan de variabele uit de graaf verdwijnen, en eindigt de tak die in het blad eindigde in de binder.

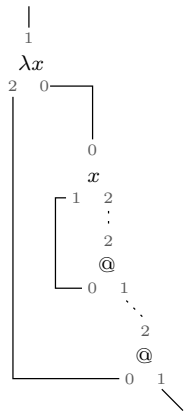


(95)

Als we afspreken dat we de variabele bij de 2-poort van de abstractor lezen, verandert

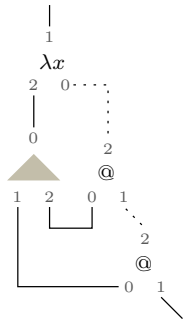
de interactie die de knoop uit de graaf haalt en het uiteinde van de tak aan de abstractor vastmaakt de expressie die we in de graaf lezen niet.

Alle variabelen die we in de takken lezen die in de 0-poort van de abstractor beginnen, bewegen in de richting van de wortel.



(96)

Zodra een volgende knoop met een gebogen tak aan de 0-poort van de abstractor vastzit, een knoop waarvan het label gelijk is aan de uitbreiding van het label van de abstractor, dan kan de interactie de gebogen tak in de 2-poort van de abstractor laten eindigen. Ze heeft hiervoor wel een waaier nodig.



(97)

Ook voor iedere volgende variabele heeft de interactie een waaier nodig: ze voegt de gebogen tak die bij die variabele hoort en de tak die in de abstractor eindigt met een waaier samen. Zo ontstaat een cascade van waaiers. Omdat iedere gebogen tak uiteindelijk in de 2-poort van de abstractor eindigt, bindt de abstractor haar variabelen lokaal.

Definitie: **lokale binding** (98)

Binding in een e-graaf is lokaal, dan en slechts dan als iedere tak waarin we een variabele lezen die een abstractor aan zich bindt, in de 2-poort van die abstractor eindigt. □

Bindt een abstractor de instanties van een variabele lokaal, dan lezen we na de interactie van een applicator en een abstractor het argument van de applicatie op de plaats van iedere instantie van de variabele. Als het argument zelf de applicator en de abstractor niet bevat tenminste.

Zit er in de 1-tak van de applicator nog een andere applicator, en volgt er daarna ook nog een abstractor, dan vind je deze na de interactie maar op één plaats terug. Omdat je ze mogelijk wel vaker leest, voorkomt lokale binding dubbel werk. Want met één interactie verdwijnen ook die applicator en de abstractor uiteindelijk uit de graaf.

We leggen de regels voor e-interactie vast: de interactie die door takken te buigen en waaiers toe te voegen de syntaxboom van een λ-expressie verandert in een graaf waarin de abstractors instanties van hun variabelen enkelvoudig en lokaal binden. Daarbij nemen we het volgende aan.

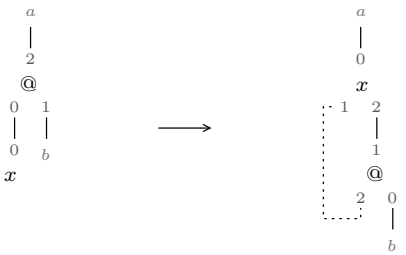
Definitie: **syntaxbomen** (99)

Interpreteren we een syntaxboom van een λ-expressie, dan lezen we in de 0-tak van een applicator een variabele of een abstractie, de functie, en in de 1-tak het argument. De abstractors in de boom hebben alleen een 0-tak. De variabelen en de wortel hebben alleen een 0-poort. □

We leggen de regels vast die lokale binding bewerkstelligen.

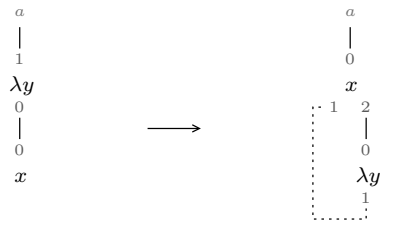
Variabelen hebben extra poorten nodig

Het zoeken naar een binder begint met de verandering van een blad in een knoop met drie poorten. Omdat een variabele het uiteinde van een tak naar de binder moet brengen, houdt ze het uiteinde van de tak vast.



(100)

De ene keer moet een variabele over een applicator springen (100), de andere keer zit een abstractor die haar niet bindt in de weg. Ook in dat geval zoekt de variabele verder.

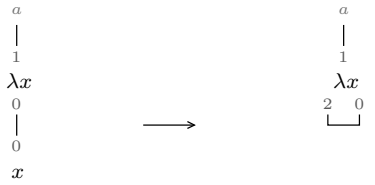


(101)

Een variabele springt alleen over een abstractor wanneer deze haar niet bindt; we kijken naar wat er gebeurt wanneer dit wel het geval is.

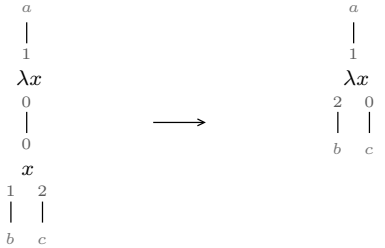
Het vastzetten van een tak

Als de abstractor die een variabele bindt er onmiddellijk aan voorafgaat, maakt interactie de binding lokaal door de variabele weg te laten en het uiteinde van de tak aan de abstractor vast te maken; de abstractor krijgt een derde poort.



(102)

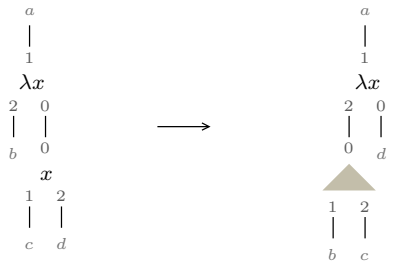
Een abstractor krijgt ook een derde poort als een variabele die verderop in de tak op zoek is een abstractor vindt die haar bindt. Het uiteinde van de tak dat die variabele meebrengt komt aan de nieuwe poort vast te zitten. De variabele verdwijnt uit de e-graaf.



(103)

Vindt een variabele een abstractor die al drie poorten heeft, dan is er een waaier nodig om

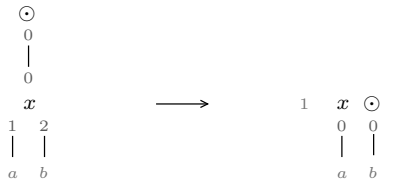
de tak die in de variabele eindigt met de abstractor te verbinden.



(104)

De waaier voegt de tak die in de 2-poort van de abstractor eindigt samen met de tak waarin we de variabele lezen. Omdat bij iedere volgende variabele die bij de abstractor hoort e-interactie nog een waaier toevoegt, kan er bij de 2-poort van de abstractor een opeenvolging van waaiers ontstaan.

Als een knoop met een variabele als label en een gebogen tak bij de wortel aankomt, bindt geen enkele abstractor in de tak die variabele. De variabele is vrij. Wanneer ze de wortel en de eerste knoop in b loslaat, komt de variabele als blad weer op haar oorspronkelijke plaats terecht.



(105)

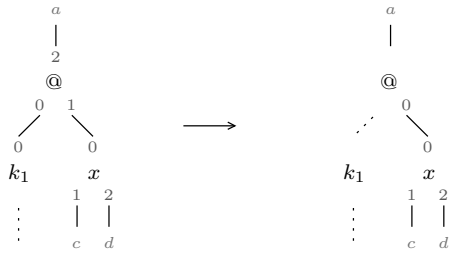
De interactie breidt het label van de variabele met 1 uit. Hierdoor kan de variabele zich niet opnieuw tussen de knopen in de graaf begeven.

De applicator blokkeert variabelen

Onderweg naar de binder springt een variabele over de knopen in de tak. Dit kan ze alleen als ze in de 0-tak van een knoop zit. Omdat een applicator twee takken, en een knoop maar één 0-poort heeft, kan een variabele, als ze de eerste knoop in een 1-tak van een applicator is, niet verder.

(106)

De variabele kan wel verder als de 1-tak in een 0-tak zou veranderen. De interactie van de applicator en de eerste knoop in b kan dit bewerkstelligen. Zien we even af van wat die interactie met die knoop en met de andere poorten van de applicator doet, dan bereikt ze in ieder geval het gewenste effect als ze dit schema volgt:



(107)

Met de interactie verandert de 1-tak van de applicator in een 0-tak. Aangezien we de variabele toch pas aan het einde van de tak lezen, verandert de interactie de expressie die we lezen niet. Wanneer we nagaan wat er op de plaats van k_1 kan staan, en hoe de poorten van k_1 en de applicator moeten veranderen, vinden de regels die variabelen verder helpen.

Waar interactie zich aan moet houden

De regels die we afleiden moeten in het schema (107) passen. Aan het schema alleen hebben we echter niet genoeg. Als er namelijk nog variabelen in de 0-tak van de applicator zitten, mag de interactie de poort waar die tak aan vastzit niet veranderen. Dit leidt tot de volgende voorwaarde.

Definitie: **e-invariant** (108)

Als we een abstractor via de 0-poort binnengaan, dan komen we in de tak geen knoop tegen die een variabele als label heeft. Als we een applicator via de 1-poort binnengaan, dan komen we in de 2-tak geen variabelen meer tegen. Als we een applicator via de 0-poort binnengaan, dan komen we in geen van beide takken nog een variabele tegen. □

Wanneer we het schema (107) invullen, mogen we veronderstellen dat de interactie de graaf vóór de stap waar maakt. Wanneer de stap zich aan de invariant houdt, en we dit voor alle schema's en regels laten zien, en de graaf waar we mee beginnen de invariant waar maakt, houdt de interactie in de graaf zich aan de invariant.

Syntaxbomen maken de invariant waar

De bomen die de syntax van λ-expressies weergeven maken de invariant in ieder geval waar.

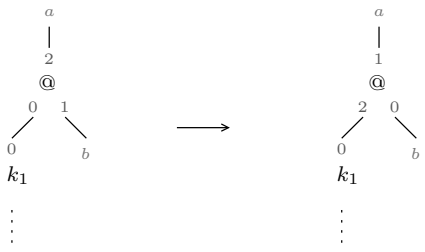
Stelling: **syntaxbomen** (109)

Een boom die de syntax van een λ-expressie weergeeft maakt de invariant waar.

Bewijs: Omdat iedere tak van een boom in een variabele eindigt, mogen we geen enkele applicator of abstractor via de 0-poort binnengaan. Vanwege de afspraak (99) doen we dit daadwerkelijk niet. □

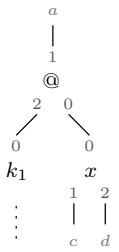
Wat we in een graaf lezen verandert niet

De invariant dwingt niet af dat je een schema zo invult dat je regels krijgt die de expressie die we in de graaf lezen niet veranderen. Zo mag de interactie een variabele niet van de ene naar de andere tak van een applicator verplaatsen. We letten hier op wanneer we regels vastleggen.



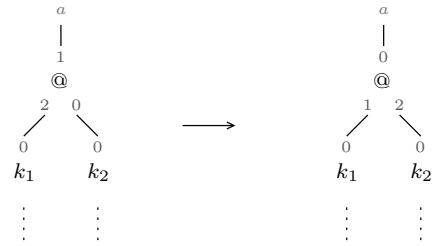
(110)

Zit k_1 met de 0-poort aan de applicator vast, dan volgen er in de tak van k_1 geen variabelen meer. De interactie kan daarmee een blokkade bij de 1-poort van de applicator opheffen: ze verandert de 0-poort in een 2-poort, en 1-poort in een 0-poort zodat x en andere variabelen in de tak verder kunnen. De 2-poort verandert in een 1-poort.



(111)

Als de nieuwe 0-tak van de applicator met een knoop begint die met haar 0-poort aan de applicator vastzit, dan volgt vanwege de invariant dat er ook in de tak geen knopen meer volgen die een variabele als label hebben. Dit betekent dat de takken die via de applicator lopen geen variabelen meer bevatten.



(112)

Omdat er geen variabelen meer volgen, kan de interactie in de graaf de poorten van de applicator veranderen, zodanig dat de applicator zelf de poorten van een knoop die er aan voorafgaat kan veranderen. Op die manier kan de applicator meehelpen aan het opheffen van een blokkade in een andere tak.

Niet alleen een applicator verandert poorten

Ook een abstractor kan de poorten van een voorafgaande knoop veranderen. Want als een abstractor met haar 0-poort aan een voorafgaande knoop vastzit, volgen er in de tak van die abstractor geen variabelen meer. Omdat een abstractor maar één tak heeft, kan alleen een knoop die ergens in de tak aan de abstractor voorafgaat nog variabelen blokkeren.

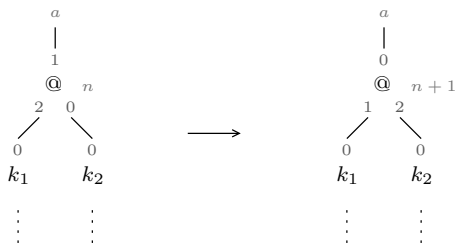
Zelfs een variabele kan meehelpen om de weg vrij te maken. Maar alleen als het een variabele betreft die zelf nog niet tussen andere knopen in een tak zit. Alleen als een tak van een applicator of abstractor uit een blad bestaat, volgen er na de variabele geen andere meer, en kan de interactie de poorten van die applicator of abstractor veranderen.

Knopen met uitgebreide labels

Als een applicator zonder variabelen in haar takken in a aan de 0-poort van een knoop k_1 vastzit, zou een volgende interactie (110) de poorten van die applicator nog een keer veranderen. Omdat die interactie echter de poorten van k_1 niet verandert, kan die knoop zelf geen voorafgaande blokkade meer opheffen.

Aangezien de interactie zich niet mag herhalen, breidt ze het label van de applicator met 1 of 2 uit. Als we daarnaast de regel alleen toepasbaar maken op een applicator waarvan de uitbreiding van het label 2 of kleiner is, dan kan een andere regel ervoor zorgen dat niet de poorten van de applicator, maar die van k_1 veranderen.

Ook in een schema waarin een abstractor de plaats van de applicator inneemt, doet het verschijnsel zich voor. De interactie die de poorten van een abstractor die geen uitgebreid label heeft verandert, breidt daarom het label met 1 uit. Het uitgebreide label geeft aan hoe vaak de interactie de poorten van de knoop heeft veranderd.



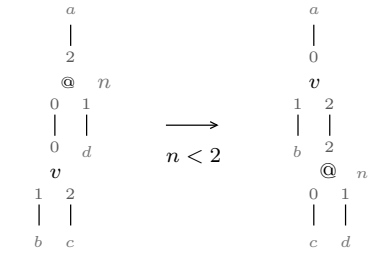
(113)

Zou het label van een applicator of een abstractor 0 als uitbreiding hebben, dan kunnen we de verandering van het label in de interactieregels in de vorm van een optelling weergeven: terwijl we regels moeten geven die het label van de abstractor en de applicator met 1 uitbreiden, doen we alsof het label van deze knopen 0 als uitbreiding heeft.

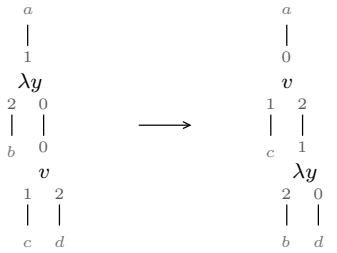
Omdat we een knoop die eigenlijk geen uitgebreid label heeft, in de regels die de interactie vastleggen vanwege de optelling een label geven met 0 uitbreiding, kom je ook in regels zonder optelling deze uitbreiding tegen; we gebruiken de uitbreiding 0 en labels zonder uitbreiding door elkaar.

Nog een afspraak over de schema's

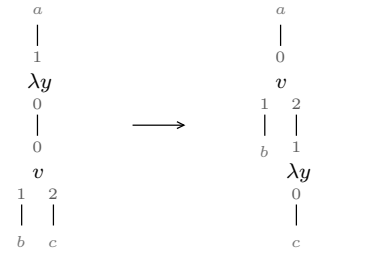
Naast het schema (113) met de applicator leggen we de regels vast die passen wanneer we de applicator door een abstractor vervangen. Daarna (p.79) volgen de regels die weergeven wat er met een variabele gebeurt die haar binder zoekt. De regels met een variabele kun je ook weer als een schema opvatten: x en y kun je door een andere naam, maar niet v vervangen.



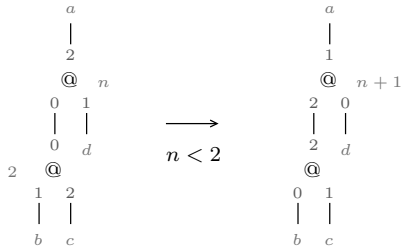
$v@$ (114)



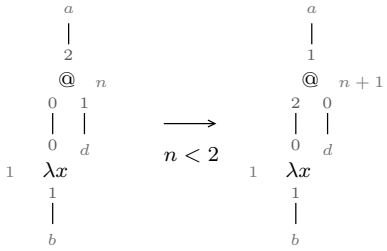
$v\lambda$ (115)



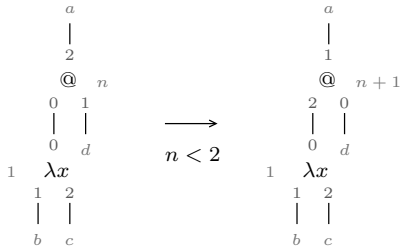
$v@$ (116)



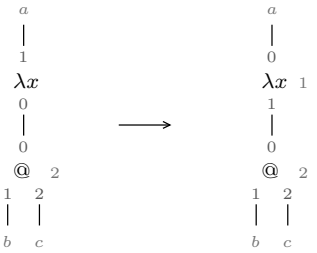
$2@$ (117)



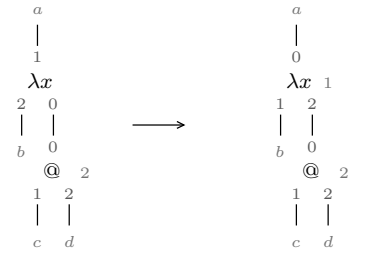
$\lambda@$ (118)



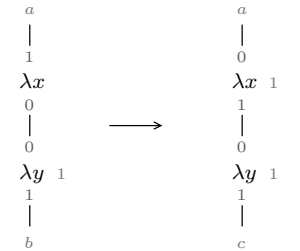
$\lambda@$ (119)



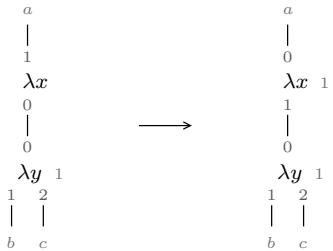
$\lambda@$ (120)



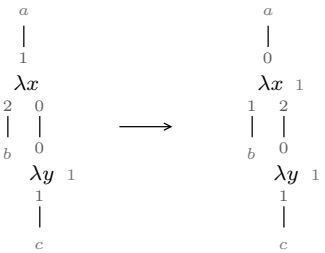
$\lambda@$ (121)



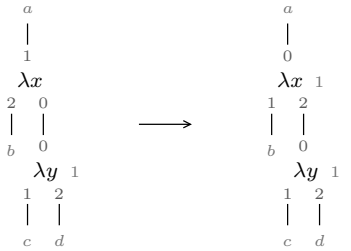
2λ (122)



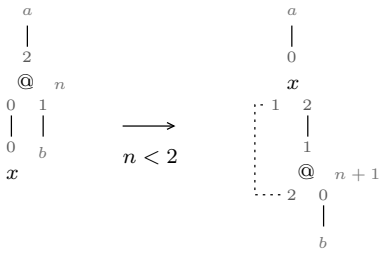
2λ (123)



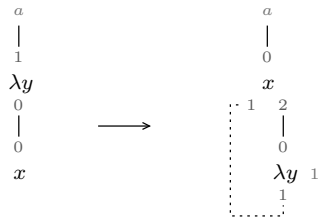
2λ (124)



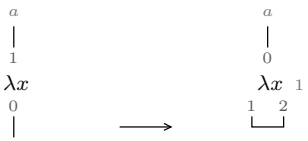
2λ (125)



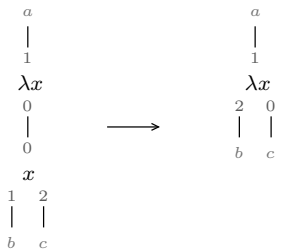
$v@$ (126)



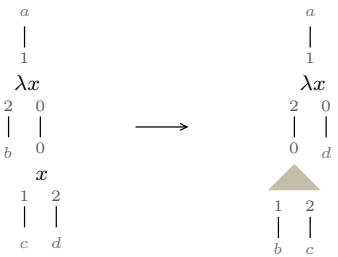
$v\lambda$ (127)



$v\lambda$ (128)



$v\lambda$ (129)



$v\lambda$ (130)

$v\odot$ (131)

Wat er verandert

Terwijl de interactie zich aan de invariant (108) moet houden omdat we anders niet kunnen laten zien dat ze de binding in de graaf lokaal maakt, moet ze, juist om lokale binding in de graaf te bewerkstelligen, ook het nodige aan de graaf veranderen.

Zo moet interactie waar een variabele aan meedoet de afstand van die variabele, tot de abstractor die haar bindt, of in ieder geval de afstand tot de wortel kleiner maken. Voegen we aan iedere volgende graaf de som van de afstanden van de variabelen tot de wortel toe, dan zouden we een functie moeten krijgen die daalt naarmate de interactie vordert.

Omdat de interactie van een variabele en de wortel, de afstand juist weer groter zou maken, tellen we alleen de afstand van variabelen zonder uitgebreid label mee. Ook bij interactie waar geen variabele aan meedoet, zou de functie niet dalen. We zien echter dat dergelijke interactie de uitbreiding van het label van één van de knopen groter maakt.

Definitie: **variante functie** (133)

Een applicator zonder uitgebreid label draagt 2 bij, een 1-applicator draagt 1 bij, een abstractor met alleen een naam in het label draagt 1 bij, en een variabele draagt de afstand tot de wortel bij. □

Wanneer we de lengte van een tak bepalen, tellen we alle knopen mee. Maar een variabele telt alleen mee als ze aan het einde van de tak zit.

Stelling: **de variante functie heeft een nulpunt** (134)

Bewijs: zoek naar de eerste knoop die geen 0-tak heeft, en ga daarbij in een applicator in de 0-tak verder. Vind je een knoop, dan wijst inspectie van de regels die we hebben vastgelegd (p.77, 78) uit dat de knoop haar 0-tak verliest, of dat de poorten van die knoop veranderen. Daarbij daalt de waarde van de functie. Omdat je het zoeken kunt herhalen draagt geen enkele knoop uiteindelijk nog bij. De variante functie heeft een nulpunt. □

Omdat de variante functie een nulpunt heeft, volgt dat iedere abstractor in de graaf haar variabelen uiteindelijk lokaal bindt.

Stelling: **lokale binding** (135)

Na interactie die de takken van de boom buigt, is de binding in een graaf lokaal.

Bewijs: omdat we in een boom die de syntax van een expressie weergeeft, in de richting (1,0) door iedere abstractor in de richting (2,0) of (2,1) door iedere applicator lopen, maakt de graaf vóórdat de interactie begint de invariant waar.

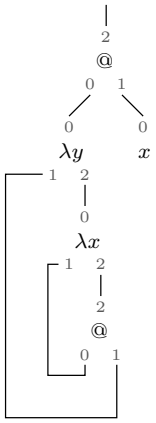
Inspectie van de regels voor de interactie wijst uit dat deze zich aan de invariant houdt, en

omdat de variante functie (134) tot nul daalt, volgt dat de graaf na interactie in ieder geval geen variabele meer bevat die bij een abstractor hoort.

Aangezien een variabele alleen bij interactie met een abstractor uit de graaf verdwijnt, en die interactie (128, 129, 130) de tak die in de variabele eindigde aan de abstractor vastmaakt, is binding na de interactie lokaal. □

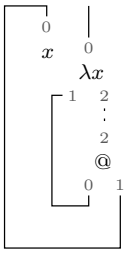
Naamloze lokale binding

Vanwege lokale binding kun je gemakkelijk het verschil zien tussen een variabele die bij een abstractor hoort, en een variabele die vrij is.



(136)

De variabele x in de 1-tak van de eerste applicator is vrij, want alle instanties van x die de tweede abstractor bindt lezen we aan het einde van de gebogen tak die in die abstractor begint. Zou de applicator echter op de abstractor inwerken, dan komt de binding in de graaf niet meer overeen met de binding in de expressie.



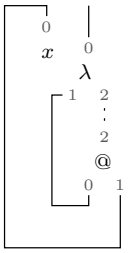
(137)

Met lokale binding, zou, omdat de 1-tak van de applicator niet in een abstractor eindigt, de instantie van x in die tak vrij zijn. Dit zie je echter niet wanneer je door de graaf loopt:

omdat je eerst de abstractor leest, bindt die zowel de instantie van x in de 1-tak als die in de 2-tak van de applicator.

Omdat een abstractor de instanties van een variabele lokaal bindt, heeft ze geen uitgebreid label nodig: kiezen we, wanneer we door een abstractor lopen, de naam van een verse variabele, en zorgen we ervoor dat we die aan het einde van de gebogen tak lezen, dan krijgen we in ieder geval een α-equivalente expressie.

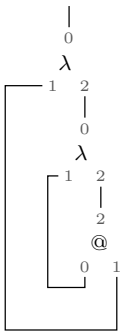
Bij de interactie van een applicator en een abstractor kan een vrije variabele die in het argument van de applicator zit gebonden raken. Maar niet als de abstractor de instanties van de variabele op een naamloze en lokale manier bindt, en je bij het lezen een naam kiest die nog niet in de graaf voorkomt.



(138)

Wanneer je in de abstractor y kiest, blijft de variabele x vrij. Met naamloze binding hoef je er niet voor te zorgen dat de naam in het label van een abstractor ergens in de graaf verandert omdat anders een vrije variabele gebonden zou raken.

Omdat we de naam van de variabele die we kiezen aan het einde van de gebogen takken van een abstractor weer nodig hebben, moeten we in die abstractor niet alleen een naam kiezen, maar deze ook onthouden: grafen met naamloze lokale binding lezen we in een context.

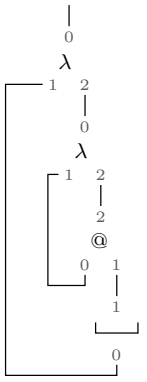


(139)

Als we de naam die we in de eerste abstractor kiezen op een stapel leggen, en bij de tweede ook, dan hoeven we aan het einde van de 0-tak van de applicator alleen maar

naar de stapel te kijken: de naam die we moeten lezen ligt bovenop. Maar aan het einde van de 1-tak zien we niet de juiste variabele.

Alleen wanneer we, zodra we van de gebogen tak van de tweede abstractor springen, de variabele die bij die abstractor van de stapel halen, lezen we wel de juiste expressie. Aan het begin van de aftakking moeten we een knoop lezen waarbij we de bovenste variabele van de stapel halen.



(140)

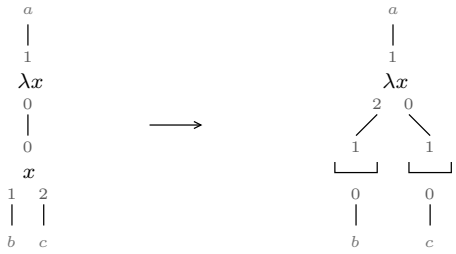
Omdat een abstractor in een λ-expressie voor wat betreft het binden van variabelen een bereik heeft, en we de variabele in de aftakking niet meer tegenkomen, noemen we de knoop die we aan het begin van de tak tegen willen komen een grens. De grens heeft een ander effect op de context dan het haakje [1, p.37] in de grafen die Asperti en Guerrini beschrijven.

De praktijk verandert

Omdat we, wanneer we een boom lezen die de syntax van een λ-expressie weergeeft, geen context nodig hebben, maar een graaf niet zonder stapel kunnen lezen, moeten we de praktijk waarin we werken uitbreiden: we hebben een extra knoop nodig, een stapel, en bewerkingen die het effect van de knoop op de stapel uitdrukken.

De grenzen komen erbij

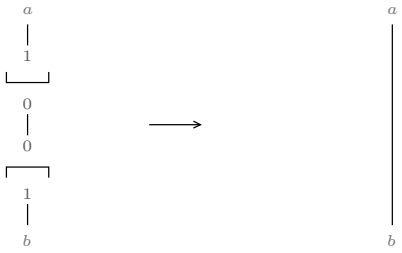
Vanwege het lokale weet de interactie in een graaf niet wanneer een aftakking begint. We laten grenzen daarom vanaf het begin en het einde van een gebogen tak naar elkaar toe bewegen: in de tak heffen ze elkaar op, en de grens die via een applicator in de aftakking terechtkomt, blijft daar.



(141)

De interactie mag grenzen toevoegen en verplaatsen, zolang we aan het einde van de gebogen tak maar de variabele lezen die bij de abstractor hoort. Hoewel de naam van de variabele uiteindelijk uit het label van de abstractor moet verdwijnen, laten we haar vanwege het lezen voorlopig nog maar even staan.

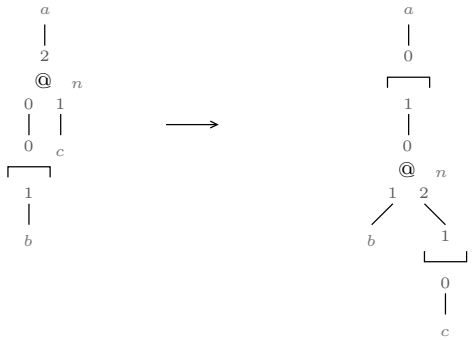
Omdat we aan het einde van een gebogen tak van een abstractor de variabele willen lezen die we wanneer we door die abstractor lopen op de stapel leggen, hebben we in die tak geen grenzen nodig. De interactie mag grenzen die elkaar in de tak tegenkomen daarom opheffen.



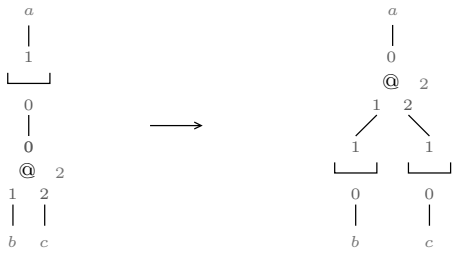
(142)

Nadert een grens de 0-poort van een applicator, dan gaan er na de interactie twee grenzen verder. Heeft het label van de applicator 0 of 1 als uitbreiding, dan nadert de grens de applicator in een tak. Na de interactie gaat er zowel in de andere tak, als in de tak die naar de applicator leidt, een grens verder.

(143)



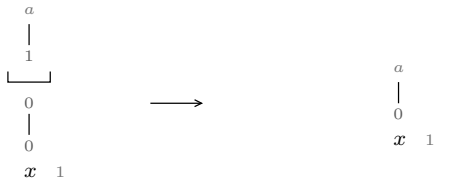
Als het label van de applicator 2 uitbreiding heeft, gaat er in beide takken een grens verder.



(144)

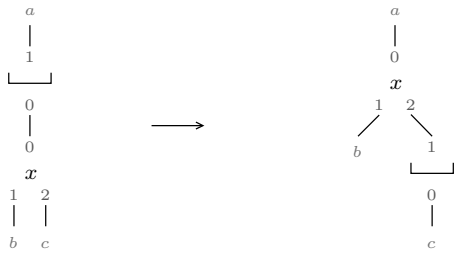
Als een grens in een gebogen tak verder gaat, heft ze een andere grens in die tak op. Omdat de tak die naar de applicator loopt deel uitmaakt van een gebogen tak, heffen de grenzen elkaar daar ook op. Als er een grens in een aftakking verdergaat, komt ze geen andere grens meer tegen. De grens geeft dan aan dat we met een aftakking te maken hebben.

Sommige takken eindigen niet in een abstractor, maar in een vrije variabele. Omdat we bij het lezen van een vrije variabele de stapel helemaal niet nodig hebben, laten we grenzen aan het einde van een dergelijke tak gewoon achterwege. De grens kan alleen het einde van de tak bereiken als de knopen in de tak naar de wortel wijzen. Ze kan dus alleen een variabele tegenkomen die een uitgebreid label heeft.



(145)

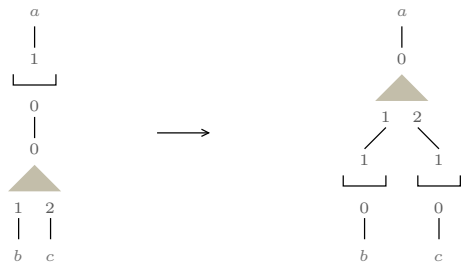
Als een variabele naar haar binder zoekt, kan ze onderweg een grens tegenkomen. Omdat het zoeken naar een binder niet van grenzen afhangt, voegen we een regel toe waarmee de interactie de variabele over de grens kan laten springen.



(146)

Omdat de grens die de interactie aan het einde van een gebogen tak toevoegt achter een cascade kan zitten, en de grens zich wel in de richting van de applicators in de tak moet

begeven, voegen we een regel toe waarmee de interactie een grens achter een waaier kan leggen.



(147)

De gebogen takken die een waaier samenvoegt beginnen in een abstractor, en daarom kan er in beide takken een grens verdergaan: de interactie van een grens en een applicator (144) laat aan de andere kant van de gebogen tak de grenzen ook in beide takken verder gaan.

Bij de interactie (143) met een applicator die wel een 0-tak heeft, gaat er ook in twee takken een grens verder: de grens in 1-tak of de 2-tak heft een grens op die van het einde van een gebogen tak komt, en de grens in de tak die naar de applicator loopt heft daar een grens op die van het begin van die tak komt.

Onderweg naar een andere grens of een aftakking, kan een grens in een gebogen tak vóór een abstractor komen te liggen. Omdat ze een andere grens op moet heffen, of in een aftakking terecht moet komen, verplaatst de interactie deze grens. Aangezien grenzen in de gebogen tak verschijnen als er geen gebonden variabelen meer in de takken zitten, volgt met de invariant (108) dat een dergelijke abstractor geen 0-tak heeft.

(148)

De interactie voegt ook in dit geval aan het begin en het einde van de gebogen tak een grens toe: in de gebogen tak of takken heffen de grenzen elkaar op, via een aftakking komt in de gebogen tak van de oorspronkelijke abstractor, en kan ze verder.

Ook voor de interactie van een grens en een abstractor die geen gebogen tak heeft, hebben we een regel nodig. We zien deze in het volgende overzicht terug. De regels in het vorige overzicht blijven van kracht.

$v\lambda$ (141)

$2\backsim$ (142)

$\backsim v$ (145)

$\backsim @$ (144)

$\backsim \triangle$ (147)

$\backsim v$ (146)

$\backsim \lambda$ (148)

$\backsim \lambda$ (149)

Grenzen en evenwicht

Als je voor iedere abstractor waar je in de gebogen tak van een andere abstractor doorheen loopt in diezelfde tak ook een grens oversteekt, houden de abstractors en grenzen die tak in evenwicht. De grenzen die zich naar een aftakking begeven zitten nog in de gebogen tak, en maken daarom geen deel uit van het evenwicht.

Definitie: **grens-gebalanceerdheid** (150)

Stelt $\triangle$ een waaier, $-$ een grens, x een variabele, en ϵ de lege expressie voor, en kun je met de regels

$$\begin{aligned} e_0 &= \lambda_{(0,2)} e_1 \lambda_{(1,0)} \\ e_1 &= \epsilon \mid @e_1 \mid \lambda x e_1 \mid \triangle e_2 \mid e_3 e_1 \\ e_2 &= \epsilon \mid \triangle e_2 \\ e_3 &= \lambda_{(0,2)} e_1 -_{(1,0)} \end{aligned}$$

de opeenvolging van labels van knopen in een gebogen tak beschrijven, dan houden de abstractors en grenzen in die tak elkaar in evenwicht. □

Omdat geen enkele clausule beschrijft dat je maar één keer, zonder er uiteindelijk weer in terug te komen, door een abstractor kunt lopen, gaat de eerste clausule altijd over één en dezelfde abstractor.

De tweede clausule beschrijft de applicators en waaiers in de tak, en het feit dat je in een tak eerst een abstractor en even later een grens die een aftakking aangeeft tegen kunt komen. De paren die uit een abstractor en een grens bestaan volgen elkaar op, of omvatten elkaar.

De derde clausule geeft aan dat als je eenmaal een waaier leest, je in een cascade zit. Na de cascade volgt geen andere knoop dan de abstractor.

Vanwege het evenwicht van abstractors en grenzen moeten de grenzen in een paar elkaar opheffen. Dit doen ze als we in iedere gebogen tak eerst een $(1,0)$-grens en later een $(0,1)$-grens lezen, en de interactie de afstand van deze grenzen verkleint.

Definitie: **e-invariant** (151)

De voorafgaande invariant (108) en het feit dat grenzen in een gebogen tak elkaar met de 0-poort aanwijzen. □

De variante functie moet ook dalen wanneer er grenzen aan de interactie meedoen. Omdat de functie (133) niet daalt wanneer de interactie grenzen toevoegt (144) leggen we een nieuwe functie vast. Draagt een abstractor ook de lengte van iedere tak bij die eindigt in de variabele die ze bindt, dan daalt de functie wel.

De grenzen mogen niet in de gebogen tak blijven zitten, en daarom draagt $(1, 0)$ - grens in een gebogen tak de afstand tot de bijbehorende $(0, 1)$ - grens aan de functie bij.

Definitie: **variante functie** (152)

Voeg naast de termen die de voorafgaande definitie (133) vastlegt, de volgende toe:

- een abstractor met een naam in het label draagt de lengte van iedere tak bij die eindigt in een instantie van de variabele die ze bindt,
- een $(1, 0)$ - grens draagt in iedere gebogen tak de afstand tot de $(0, 1)$ - grens bij of, als we die niet passeren: de afstand tot de vrije variabele aan het einde van de tak.

Wanneer we een afstand of een lengte bepalen, tellen we waaiers niet mee. □

Stelling: **de variante functie heeft een nulpunt** (153)

Bewijs: de redenering in het bewijs van de oorspronkelijke stelling (134) werkt ook met de nieuwe regels (p.87) en de nieuwe variante functie: inspectie van de regels wijst uit dat de grenzen in een gebogen tak elkaar naderen en opheffen.

Wanneer de interactie een tak van de abstractor met de cascade verbindt, telt de lengte van die tak niet meer mee. Omdat die interactie een waaier toevoegt, telt de afstand tussen de grenzen één keer extra mee. Aangezien de tak vanwege de variabele één langer is, en we waaiers niet meetellen, daalt de waarde van de functie toch. □

Stelling: **gebalanceerdheid** (154)

Breiden we de regels voor de interactie die lokale binding bewerkstelligt uit met de regels die betrekking hebben op grenzen, dan houden abstractors en grenzen elkaar uiteindelijk in evenwicht.

Bewijs: omdat de variante functie daalt en een nulpunt heeft (153) bewegen de grenzen in een gebogen tak van een abstractor naar elkaar toe. Inspectie van de regels wijst uit dat wanneer een grens bij een applicator komt, er twee grenzen verder gaan.

Als een grens in een tak van de applicator verder gaat, een tak die in een abstractor eindigt, dan heft de grens vanwege de invariant de andere grens in die tak op. Als de tak in een variabele eindigt, geeft de grens het begin van een aftakking aan.

Als een grens verder gaat in de tak die naar de applicator leidt, dan heft ze daar, ook weer vanwege de invariant, de andere grens in de tak op. □

De binding blijft lokaal

Stelling: **lokale binding** (155)

Ook met inbegrip van de regels voor grenzen bewerkstelligt de interactie lokale binding.

Bewijs: wanneer we de regels (p.87) voor de interactie met grenzen toevoegen, kunnen we, als we haar op de nieuwe invariant en de nieuwe variante functie betrekken, de redenering in het bewijs van de voorafgaande stelling (135) volgen. □

Wat er nog moet gebeuren

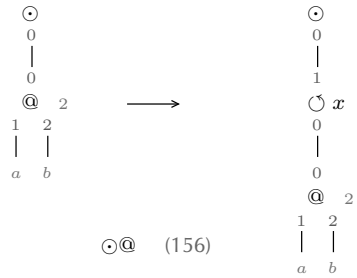
Omdat het label van een abstractor de naam van de variabele bevat, hebben we nog geen naamloze binding. Wanneer we door een abstractor lopen zouden we ook kunnen doen alsof het label geen naam bevat, een verse naam op de stapel leggen, en aan het einde van de tak alleen naar de stapel kijken. We kunnen door interactie met een nieuwe knoop echter het label uit de naam halen.

Na het bewerkstelligen van lokale binding wijzen alle knopen met hun 0-poort naar de wortel. Dit betekent dat als de 1-tak van een applicator met een abstractor begint, er geen optimale stap kan volgen omdat beide knopen niet op elkaar in kunnen werken. De interactie moet de poorten van de applicator dus nog veranderen, en wel zodanig dat de applicator en de abstractor wel kan volgen.

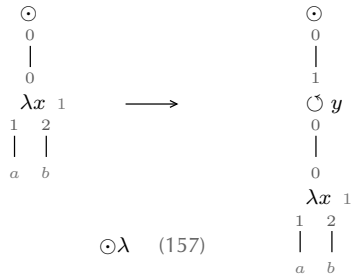
We voegen een nieuwe knoop toe die de rest van de interactie niet in de weg zit, een knoop die vanaf de wortel van de graaf de takken in gaat en het nodige in de graaf verandert: een abstractor krijgt λ als label, het uitgebreide label in de applicator verdwijnt, en haar poorten van de applicator draaien tegen de klok in één op. Met grenzen hoeft er niets te gebeuren.

Wanneer een vrije variabele bij de wortel komt, voorkomt de interactie (131) door het label met 1 uit te breiden, dat de variabele weer naar een binder gaat zoeken. Als de nieuwe knoop aan het einde van de tak bij de variabele komt, mag ze het label van de variabele dus niet veranderen. We zorgen later voor interactie die de uitbreiding uit het label haalt.

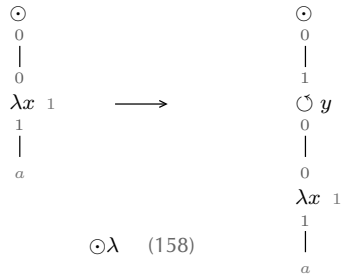
De nieuwe knoop die de poorten en labels van andere knopen moet veranderen geven we $\circlearrowleft$ als label. Vanwege de lokale binding wijst iedere applicator en iedere abstractor met haar 0-poort naar de wortel. De eerste knoop in de tak van de wortel zit dus met de 0-poort aan de wortel vast. We leggen daarom zowel voor de interactie van de wortel en de applicator, als die van de wortel en de abstractor regels vast die een $\circlearrowleft$ toevoegen.



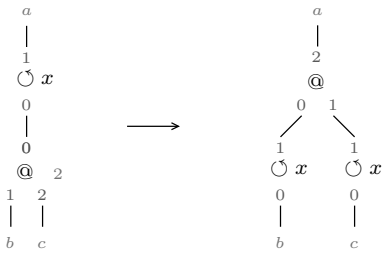
⊙@ (156)



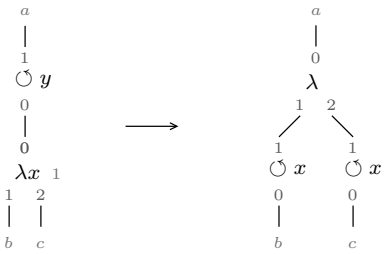
⊙λ (157)



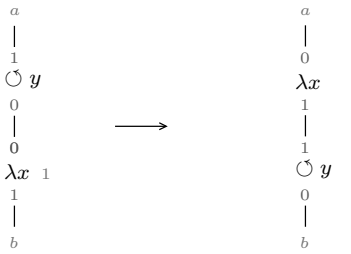
⊙λ (158)



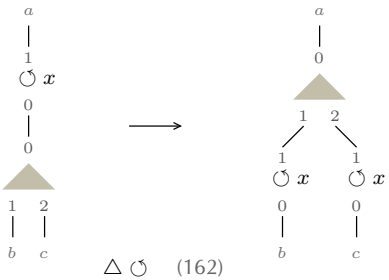
@ ↺ (159)



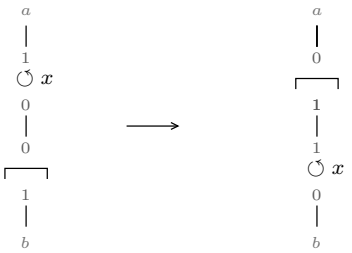
λ ↺ (160)



λ ↺ (161)



△ ↺ (162)



⌙ ↺ (163)

↻↻ (164)

$v\odot$ (165)

De ↺-knoop verschijnt met een willekeurig uitgebreid label in de graaf. Komt ze een abstractor tegen, dan raakt deze de naam van de variabele kwijt. De nieuwe knoop neemt de naam in haar uitgebreide label mee. Op die manier zorgt ze ervoor dat we, ondanks de grenzen die eventueel nog in de gebogen tak zitten, aan het einde van die tak toch juiste de juiste variabele lezen.

Zodra interactie lokale binding bewerkstelligt wijst iedere knoop met haar 0-poort in de richting van de wortel. De ↺-knoop die bij de wortel in de graaf verschijnt kan daarom op alle knopen in de graaf inwerken. Omdat er bij een applicator in beide takken een ↺-knoop verder gaat, komt er in iedere tak een ↺-knoop terecht.

Als een abstractor een gebogen tak heeft, gaat er zowel aan het begin als aan het einde van die tak een ↺-knoop verder. Zit een ↺-knoop aan het einde van een gebogen tak, dan springt deze in ieder geval over de waaiers, want die wijzen met hun 0-poort naar het einde van de tak.

De knopen in een gebogen tak van een abstractor heffen elkaar in ieder geval op als ze elkaar aanwijzen. We leggen dit vast door de invariant uit te breiden.

Definitie: **e-invariant** (166)

De voorafgaande invariant (151) en het feit dat als er een ↺-knoop in de gebogen tak van een abstractor zit, die tak ook een tweede ↺-knoop bevat, en dat we eerste ↺-knoop in de richting (1,0) en de tweede in de tegenovergestelde richting doorlopen. Als er zich een paar grenzen in de gebogen tak bevindt, dan omvatten de ↺-knopen deze. □

De interactie moet de knopen in de tak daadwerkelijk dichter bij elkaar brengen. Daarom breiden we ook de variante functie uit. Daarbij kijken we niet alleen naar de ↺-knopen zelf, maar ook naar termen die andere knopen, vanwege interactie met een dergelijke knoop aan de waarde van de functie bijdragen.

Omdat we de afstand van een ↺-knoop tot een andere knoop mee willen tellen, en er in iedere tak van een applicator een ↺-knoop verder gaat, laten we ook de lengte van iedere tak waar de ↺-knoop in zit, meetellen. Zouden we dit niet doen, dan daalt de waarde van de functie bij die interactie met een ↺-knoop niet.

Definitie: **variante functie** (167)

Voeg naast de termen die de voorafgaande definitie (152) vastlegt ook bijdragen van de wortel en de ↺-knoop toe, en voeg een voorwaarde aan de bijdrage van een applicator zonder uitgebreid label toe:

- als ze een 0-poort heeft draagt de wortel één plus de lengte van alle takken bij,
- een ↺-knoop draagt de afstand van de wortel en de afstand van de knoop zelf tot

het einde bij, of als er twee ↺-knopen in een tak zitten, de afstand van de ene tot de andere,

- een applicator zonder uitgebreid label draagt alleen bij als er alleen variabelen met een 0-poort in haar takken zitten.

Wanneer we de afstand van de wortel en het einde van een tak, de afstand van de ene en de andere ↺-knoop, of de afstand van een ↺ en het einde van een tak bepalen, tellen waaiers en grenzen niet mee. □

We stellen een voorwaarde aan de bijdrage (133) van een applicator die geen uitgebreid label heeft uit omdat de interactie met een ↺-knoop de poorten van de applicator zodanig verandert dat de variante functie weer zou stijgen.

Stelling: **de variante functie heeft een nulpunt** (168)

Bewijs: met de nieuwe regels (p.91, 92) en de nieuwe variante functie (167) kunnen we de redenering in het bewijs van de voorafgaande stelling (153) volgen: inspectie van de regels wijst uit dat de ↺-knopen in een gebogen elkaar naderen en opheffen, een en ↺-knoop in een tak die in een variabele eindigt, zich naar die variabele kan bewegen. □

De abstractors en grenzen blijven in evenwicht

De interactie waar een ↺-knoop aan meedoet mag het evenwicht (150) van waaiers en grenzen niet verstoren.

Stelling: **gebalanceerdheid** (169)

Ook wanneer we regels voor de interactie met ↺-knopen toevoegen houden abstractors en grenzen elkaar uiteindelijk in evenwicht.

Bewijs: de redenering in het bewijs van de voorafgaande stelling (154) werkt ook met de nieuwe regels en de nieuwe variante functie: de variante functie heeft een nulpunt, en met de inspectie van de regels volgt dat een ↺-knoop de interactie van de andere knopen niet in de weg zit. □

Eindelijk naamloze lokale binding

Omdat de ↺ de naam van de variabele uit het label van de abstractors haalt, bewerkstelligt interactie eindelijk naamloze lokale binding, tenminste als de binding ook met interactie waar ↺-knopen aan meedoen lokaal blijft.

Stelling: **lokale binding** (170)

Ook met inbegrip van de interactieregels voor ↺-knopen bewerkstelligt e-interactie lokale binding. Bovendien heeft in een e-normaalvorm geen enkele abstractor die een instantie van een variabele bindt nog de naam van die variabele in haar label.

Bewijs: de redenering in het bewijs van de in de voorafgaande stelling (155) werkt ook wanneer we de regels (p.91, 92) voor interactie met ↺-knopen toevoegen, en haar op de nieuwe invariant en nieuwe variante functie betrekken. Omdat de variante functie een nulpunt heeft, dragen ook de abstractors niet meer bij. Vanwege de definitie van de variante functie kan dit alleen als het label geen naam meer bevat. □

Alle regels bij elkaar

De regels die we hebben afgeleid veranderen syntaxbomen in grafen met naamloze lokale binding. Vóórdat we de interactie bekijken die zich in deze grafen richt op het zetten van optimale stappen, leggen we de interactie vast die naamloze lokale binding bewerkstelligt.

De regels die we hebben afgeleid veranderen syntaxbomen in grafen met naamloze lokale binding. Vóórdat we de interactie bekijken die zich in deze grafen richt op het zetten van optimale stappen, leggen we door de structuren (49, 55) in te vullen, de interactie vast die naamloze lokale binding bewerkstelligt.

Definitie: **e-regels** (171)

De schema's (p.77, 78, 79, 87, 91, en 92) leiden tot de regels voor de interactie in een e-graaf. □

Definitie: **e-interactienet** (172)

Met

- V de verzameling variabelen,
- $\mathcal{L}$ de verzameling

$$\begin{aligned}&\{\lambda, \langle\lambda, x\rangle, \langle\lambda, x, n_0\rangle,\\ &@, \langle @, n_1\rangle,\\ &x, \langle x, n_0\rangle,\\ &-, \triangle, \odot, \circlearrowleft\, x\\ &\mid x \in V, n_0 \in \{0,1\}, n_1 \in \{0,1,2\}\}\end{aligned}$$

· $\alpha \subset \mathcal{L} \times \mathbb{N}$ de relatie

$$\{\langle \lambda, 3\rangle, \langle \lambda x, 2\rangle, \langle\langle \lambda x, n_0\rangle, 2\rangle, \langle\langle \lambda x, n_0\rangle, 3\rangle,$$
$$\langle @, 3\rangle, \langle\langle @, n_1\rangle, 3\rangle,$$
$$\langle x, 1\rangle, \langle x, 3\rangle, \langle\langle x, n_0\rangle, 1\rangle,$$
$$\langle -, 2\rangle, \langle \triangle, 3\rangle, \langle \odot, 1\rangle, \langle \circlearrowleft x, 2\rangle$$
$$\mid x \in V, n_0 \in \{0, 1\}, n_1 \in \{0, 1, 2\}\}$$

· g een $\mathcal{L}\alpha$-graaf, en

· R de verzameling e-regels,

is een e-interactienet het geordende paar $\langle g, R\rangle$. □

Stelling: **robuustheid** (173)

De definitie van e-interactienet is robuust.

Bewijs: R is deterministisch en voor iedere $r \in R$ is $\Pi_3(r)$ een bijectie. □

Definitie: **e-interactie** (174)

Met b een boom (99) die de syntax van een λ-expressie weergeeft, en R de verzameling e-regels, is e-interactie in het net $\langle b, R\rangle$ de opeenvolging van stappen die met een stap

$$\langle\langle b, R\rangle, \langle g, R\rangle\rangle$$

begint. □

Een voorbeeld van e-interactie

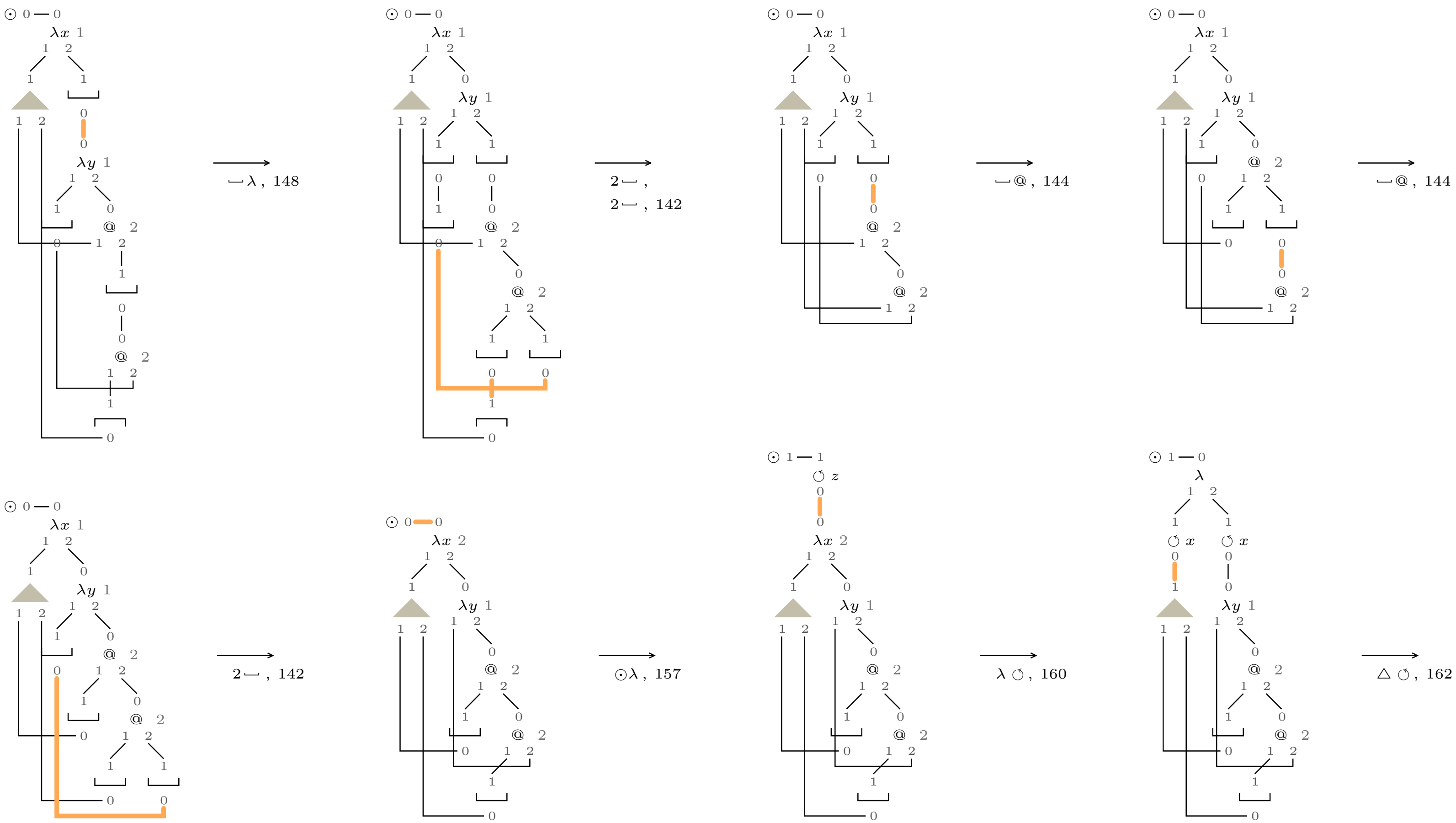
We hebben de nodige regels afgeleid voor de interactie die bomen in grafen verandert. Bij wijze van voorbeeld kijken we hier naar de interactie die een syntaxboom die de Church-weergave

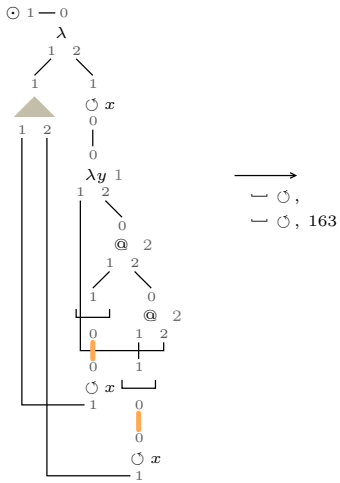
$$\lambda x \lambda y.x(xy)$$

van het natuurlijke getal 2 in een graaf met naamloze lokale binding verandert.

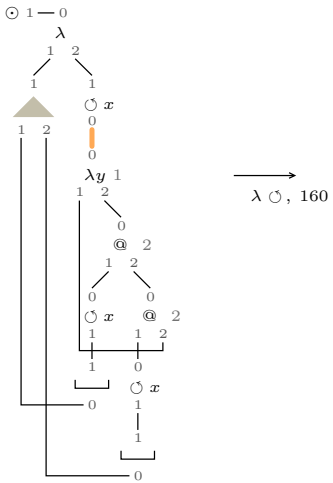
⊙ 0 — 1
λx
λy
@
x
x y
v@ , 114
v@ , 114
v@ , 114
vλ , 115
vλ , 129
v@ , 114
vλ , 116
vλ , 130

v@, 114
vλ, 129
2@, 117
λ@, 121
2λ, 132
⌙△, 147
⌙@, 144
2⌙, 142

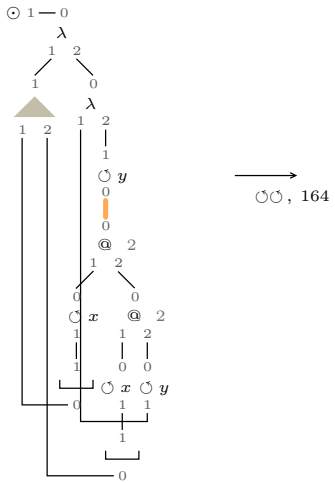

⎵λ, 148
2⎵,
2⎵, 142
⎵@, 144
⎵@, 144
2⎵, 142
⊙λ, 157
λ ↺, 160
△ ↺, 162



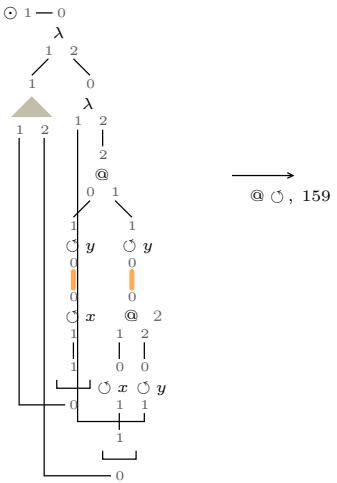
→ ⍉ ,
⍉ ⍉ , 163



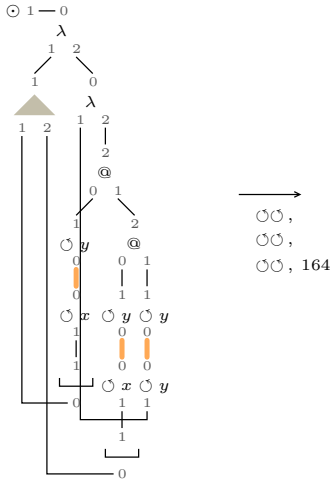
→ λ ⍉ , 160



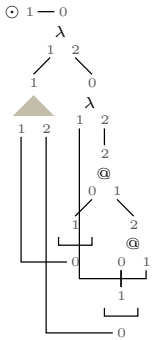
→ ⍉⍉ , 164



→ @ ⍉ , 159



→ ⍉⍉ ,
⍉⍉ ,
⍉⍉ , 164



Volledigheid

De interactie moet een boom die de syntax van een λ-expressie weergeeft veranderen in een graaf waarin interactie optimale stappen kan zetten. Dit betekent dat als een applicatie met een abstractie begint, de applicator en de abstractor op elkaar in moeten kunnen werken.

Stelling: **reductie** (175)

Als de tak van een applicator met een abstractor begint, dan zitten de applicator en de abstractor met hun 0-poort aan elkaar vast.

Bewijs: omdat de variante functie een nulpunt heeft, telt uiteindelijk geen enkele applicator of abstractor meer mee. Dit betekent een applicator een 0-tak en een 1-tak heeft, en dat er in een abstractor alleen een 2-tak verder gaat. Als een applicator aan een abstractor vastzit, dan verbindt een kant hun 0-poorten. □

Deze eigenschap kun je als de volledigheid van e-interactie kan zien. Het is namelijk het enige dat die interactie kan doen om ervoor te zorgen dat een applicator die een abstractor in haar 0-tak heeft op die abstractor in kan werken.

Er ontbreekt nog één e-eigenschap

Hoewel we er bij het afleiden van de regels voor de interactie wel rekening mee hebben gehouden, hebben we nog niet laten zien dat interactie de λ-expressie die we in een graaf lezen niet verandert; we moeten nog laten zien dat e-interactie correct is.

Een waaier die takken splitst

Vanwege interactie in e-grafen behelst de context waarin we lezen niet meer dan een stapel met namen van variabelen. Omdat we bij interactie met optimale stappen grafen krijgen waarin we in de tegenovergestelde richting door een waaier lopen, moeten we de context uitbreiden. We gaan in onderdeel 4 bovendien na welke knopen we nodig hebben om de context zo te laten veranderen dat we steeds de juiste expressie lezen.

4 Waaiers en context

Met het buigen van de takken zorgt de interactie in een e-graaf ervoor dat een abstractor de verschillende instanties van de variabele enkelvoudig en lokaal bindt. Omdat de interactie de binding ook naamloos maakt, hoeft de interactie die optimale stappen zet namen niet uniek te maken, en verdwijnen de applicator en abstractor met één enkele optimale stap.

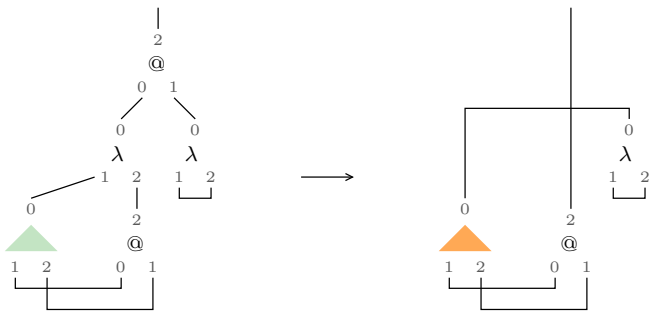
Aangezien abstractors geen uitgebreid label meer hebben, moeten we wanneer we door een graaf lopen bij een abstractor een verse variabele kiezen en onthouden, want aan het einde van een gebogen tak lezen we de naam die we bij de abstractor kiezen. Vanwege de interactie in een e-graaf lezen we in een context.

We lezen echter niet alleen vanwege variabelen in een context. Want de waaiers die met het buigen van de takken in de graaf verschijnen zitten soms de interactie van een applicator en een abstractor in de weg. Wanneer de interactie ze verplaatst, moeten we ook de manier waarin we door waaiers lopen onthouden.

We gaan na hoe de interactie met waaiers eruitziet, welke vorm de context heeft, en op welke manier de knopen in een graaf de context veranderen. We geven ook aan waar interactie die optimale stappen zet zich aan moet houden opdat het lezen van een graaf in die context werkt.

Waaiers die in de weg zitten

Met het buigen en samenvoegen van de takken verschijnt er, zodra twee takken in dezelfde abstractor moeten eindigen, een waaier in de graaf. Oorspronkelijk zit die waaier met haar 0-poort aan de 1-poort van de abstractor vast. Verdwijnt de abstractor, dan zit de waaier niet langer aan het einde van de tak.



(176)

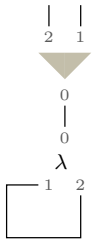
De waaier bevindt zich tussen de knopen in de graaf. Ze gaat daar in de 1-tak van de applicator een abstractor vooraf. Dit betekent dat we na de interactie wel een reduceerbare

expressie lezen, maar de applicator en de abstractor niet op elkaar in kunnen werken omdat de waaier in de weg zit.

Omdat we de waaier zelf niet in de expressie terugzien, maakt het niet uit waar de waaier zich bevindt, zolang we maar dezelfde expressie lezen. We zoeken daarom naar interactie, zodanig dat de waaier plaats maakt voor de interactie van de applicator en de abstractor.

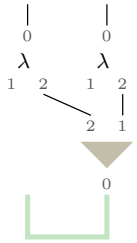
De interactie van een waaier en een abstractor

De waaier (176) zit tussen de applicator en de abstractor in, en wat het plaats maken betreft maakt het niet uit waar ze terechtkomt. Omdat de waaier met de 0-poort aan de 0-poort van de abstractor vastzit, ligt interactie met de abstractor voor de hand. De waaier maakt alleen plaats als de interactie de volgorde van de waaier en de abstractor verandert.



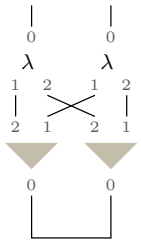
(177)

De graaf mag veranderen, maar alleen op een zodanige manier dat de expressie die we er in lezen niet verandert. Omdat we in beide takken die via de waaier lopen een abstractor lezen, moeten we na de interactie in beide takken, nog voordat we de waaier tegenkomen, eerst een abstractor lezen.



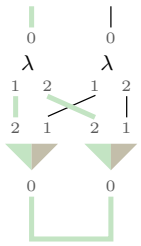
(178)

De expressie die we lezen verandert alleen dan niet als vervolgens ook de gebogen tak van de abstractor hetzelfde blijft. Omdat interactie lokaal is, verandert ze die tak niet. Na de laatste knoop in de tak moeten we een variabele lezen, en wel dezelfde als we bij de abstractor hebben gekozen. Dit betekent dat de tak zich moet splitsen.



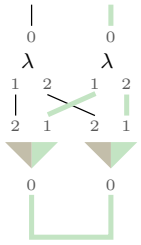
(179)

Ook voor het splitsen van een tak gebruiken we een waaier: we spreken af dat wanneer we de 0-poort van een waaier naderen we in de 1-poort of in de 2-poort verder gaan met lezen. Niet in beide, zoals bij de applicator, maar in één van de twee. Omdat de abstractor kan verdwijnen, onthouden we niet de abstractor zelf, maar de poort waardoor we de eerste waaier binnengaan.



(180)

Verlaten we de tweede waaier in het paar via de 2-poort dan lezen we niet alleen de variabele die we bij de abstractor hebben gekozen, de gebogen tak eindigt ook waar ze begint, en daardoor blijft de binding lokaal. Wanneer we de andere abstractor lezen, onthouden we de poort op dezelfde manier.



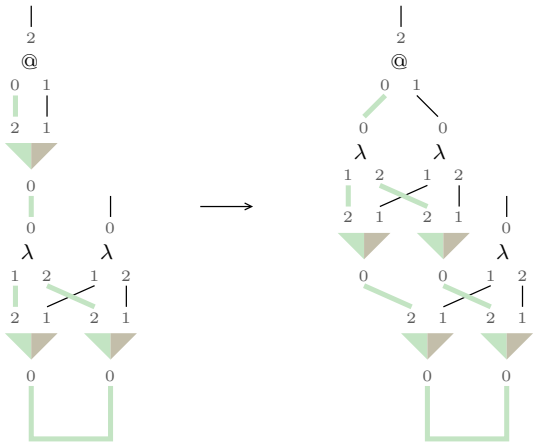
(181)

Definitie: **de richting voor het lezen van waaiers in een paar** (182)

Als we de eerste waaier in een paar dat door interactie van een waaier en een abstractor ontstaat via de n-poort binnengaan, dan verlaten we de tweede waaier in dat paar via de n-poort. □

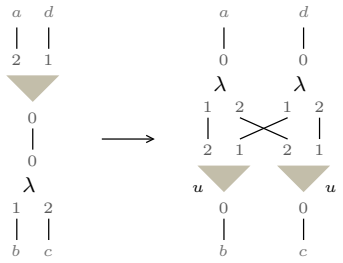
Waaiers houden elkaar in evenwicht

Met de interactie (177) van een waaier en een abstractor komt er een paar waaiers in de gebogen tak terecht. Gaat nog een waaier aan één van de abstractors vooraf, dan komt er een tweede paar waaiers in de gebogen takken van de abstractors terecht. Het nieuwe paar omvat het oude.



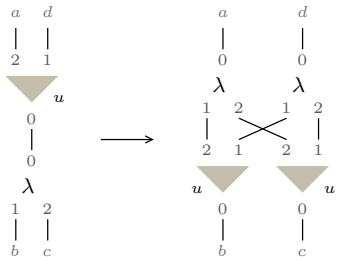
(183)

Omdat het ene paar waaiers het andere omvat, kun je de namen van de poorten onthouden door ze op een stapel te leggen. Zonder dat we de hele graaf overzien, en dat doen we niet wanneer we lezen, weten we niet of een waaier waar we doorheen lopen een paar vormt met een andere waaier. We laten de interactie daarom het label van een waaier in een paar met u uitbreiden.



(184)

Als een waaier een paar vormt met een andere, en op een volgende abstractor inwerkt, vormt die waaier niet alleen in de gebogen tak van die abstractor maar ook in de tak van de voorafgaande abstractor een paar met een andere waaier. Bij de interactie houdt de waaier haar uitgebreide label.



(185)

Omdat de gebogen tak van een abstractor na de interactie van die abstractor met een waaier begint en eindigt, houden de waaiers die de aanduiding u in het label hebben elkaar in die tak in evenwicht.

Definitie: **waaier-gebalanceerdheid** (186)

Stelt $\triangle$ een waaier zonder uitgebreid label, $\triangle u$ een waaier met een uitgebreid label, x een variabele, $-$ een grens, en ϵ de lege expressie voor, en kun je met de regels

$$\begin{aligned} e_0 &= \epsilon \mid @e_0 \mid \triangle e_0 \mid \lambda e_0 \mid \lambda x\, e_0 \mid -e_0 \mid e_1 e_0 \\ e_1 &= \triangle u_{(n,0)} e_0 \triangle u_{(0,n)} \end{aligned}$$

de opeenvolging van labels van knopen in een tak beschrijven, dan houden de waaiers met een uitgebreid elkaar in die tak in evenwicht. □

Omdat je, nadat je door een waaier die 1 in haar label heeft van de tak kunt springen, maken waaiers geen deel uit van het evenwicht van abstractors en grenzen. We zien die knopen, net als de applicator daarom als losse knopen in de definitie terug. De tweede clausule beschrijft waaiers die een paar vormen.

Als de gebogen tak door interactie van een applicator en een abstractor verdwijnt, kan het ene evenwicht ook op het andere volgen. De definitie beschrijft dit met de laatste term in de eerste clausule.

De interactie in de graaf moet het evenwicht van waaiers in stand houden; we laten de waaier-gebalanceerdheid van takken takken deel uit laten maken van de o-invariant.

Grafen lezen met een stapel

Als we ons aan de afspraak over waaiers (182) willen houden, hebben we, wanneer we de tweede waaier in een paar verlaten, de naam van de poort nodig waardoor we de eerste waaier in het paar binnengaan. Vanwege het evenwicht van waaiers in de gebogen takken

van een graaf zouden we een stapel kunnen gebruiken om de namen van de poorten te onthouden.

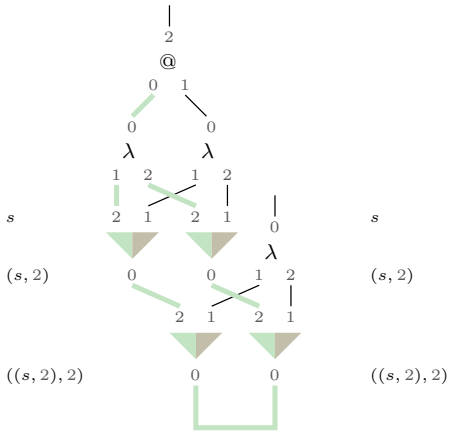
Definitie: **stapels met namen van poorten** (187)

S, de verzameling stapels, leggen we door middel van de volgende clausules vast:

$$\begin{array}{lllll} & & & & \epsilon \in S \\ p \in \{1,2\} & & & \rightarrow & (p) \in S \\ p \in \{1,2\} & \wedge & s \in S \quad \wedge \quad s \neq \epsilon & \rightarrow & (s,p) \in S \end{array}$$

□

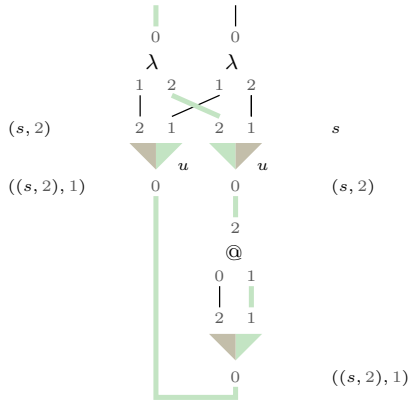
Het lezen met een stapel gaat als volgt. Wanneer we de eerste waaier in een paar via de n-poort binnengaan, dan leggen we n op de stapel. Vanwege de gebalanceerdheid van waaiers ligt, wanneer we daarna bij de tweede waaier in hetzelfde paar komen, n boven op de stapel. We halen n van de stapel, en verlaten de waaier via de n-poort.



(188)

Als de tak tussen de tweede en bijbehorende waaier een gebalanceerde tak is, liggen bij de bijbehorende waaier dezelfde namen op de stapel als na de tweede waaier. Wanneer we voor wat betreft de keuze van de poorten naar de stapel kijken, houden we ons dus aan de afspraak (182) voor het doorlopen van de waaiers in een graaf.

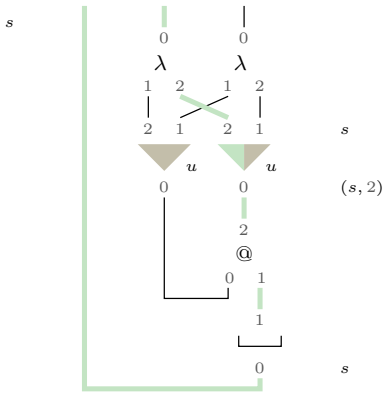
Als we tussen twee waaiers die een paar vormen door een losse waaier lopen, en daarbij de naam van een poort op de stapel leggen, dan ligt die naam, juist omdat deze bij een losse waaier hoort, wanneer we bij de tweede waaier in het paar komen, nog steeds op de stapel. Hierdoor kiezen we in de tweede waaier mogelijk de verkeerde poort.



(189)

Omdat een losse waaier geen paar vormt met een andere, hoeven we echter helemaal geen naam te onthouden. Wanneer we bij een losse waaier, een waaier die geen uitgebreid label heeft, niets met de stapel doen, dan kunnen we voor wat betreft de keuze van de poort in de tweede waaier van een paar op de stapel vertrouwen.

Spring je na een waaier van de tak waarin later de enige waaier waarmee deze een paar vormt volgt, dan zou je, net als bij een losse waaier, eigenlijk niets op de stapel mogen leggen. Wanneer je echter door de waaier loopt, weet je nog niet dat je in een aftakking verder gaat, en daardoor de tweede waaier in het paar niet meer bereikt.



(190)

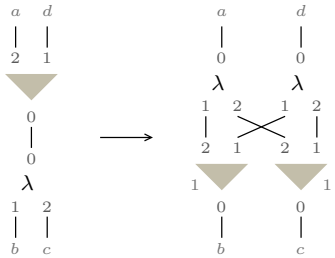
Omdat je bij de waaier niet weet of je de naam van de poort moet onthouden, zit er niets anders op dan de naam op de stapel te leggen. Want aan het uitgebreide label (185) van de waaier kun je niet zien of die buiten de tak die in de abstractor begint een paar vormt met een andere waaier.

Wanneer je de grens oversteekt, en de waaier geen paar vormt met een andere waaier in de tak die na de grens volgt, moet je bij de grens de naam van de poort weer van de stapel

halen. De waaier vormt buiten de gebogen tak geen paar met een andere als het paar waaiers uit de interactie van de abstractor en een losse waaier (184) voortkomt.

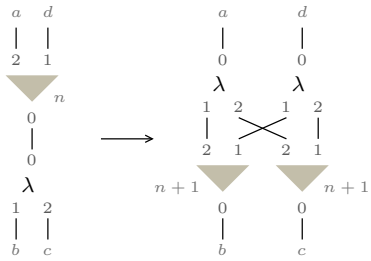
Waaiers horen dus in zeker opzicht bij een abstractor: als een waaier zonder uitgebreid label op een abstractor inwerkt begint de gebogen tak van die abstractor met een waaier die buiten die tak geen paar vormt met een andere waaier. Het paar waaiers hoort bij de gebogen tak, en daarmee ook bij de abstractor.

We laten de interactie daarom het label van een waaier niet met u maar met een natuurlijk getal uitbreiden. Als een waaier een 1 in haar label heeft, dan hoort die waaier bij de voorafgaande abstractor, dat wil zeggen: vormt ze alleen in een tak die in die abstractor begint en eindigt een paar met een andere waaier.



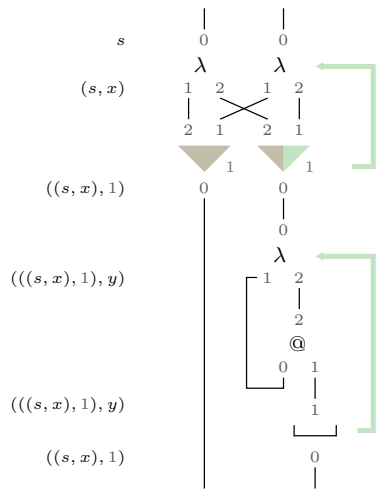
(191)

Als een waaier met 1 in haar label op een abstractor inwerkt, vormt ze ook buiten de tak die in die abstractor begint een paar met een andere waaier. We laten de interactie daarom één bij de uitbreiding van het label van een waaier optellen. Zo weten we bij de grens dat we de poort die bij de waaier hoort op de stapel moeten laten liggen.



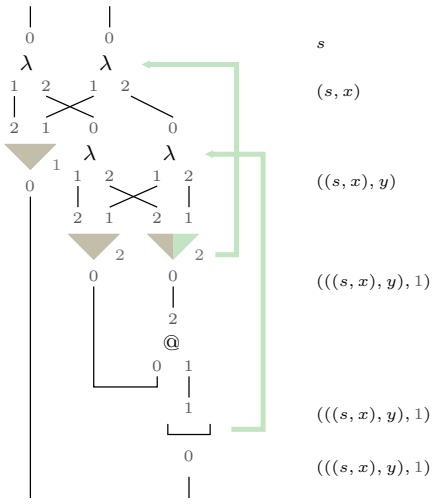
(192)

Met de nieuwe uitbreiding van het label (191, 192) vind je de namen die we bij een grens van de stapel moeten halen. Zoek, om te beginnen, bij de grens naar het begin van de tak waar je vanaf springt. De namen van poorten die bij een waaier in die tak horen, moeten, als de waaier buiten de tak geen paar vormt met een andere, van de stapel verdwijnen.



(193)

Omdat er in de gebogen tak van de tweede abstractor geen waaier zit, hoef je bij de grens ook geen naam van de stapel te halen. We kijken niet verder dan de abstractor die bij de grens hoort. De naam van de poort die we bij de waaier in de gebogen tak van de eerste abstractor onthouden, blijft gewoon liggen.



(194)

Werkt de waaier op de tweede abstractor in, dan verschijnt er een waaier in de gebogen tak van die abstractor. Omdat die waaier niet bij de tweede maar bij de eerste abstractor hoort, blijft de naam van de poort bij de grens gewoon op de stapel liggen: de waaier vormt buiten de gebogen tak van de tweede abstractor een paar met een andere.

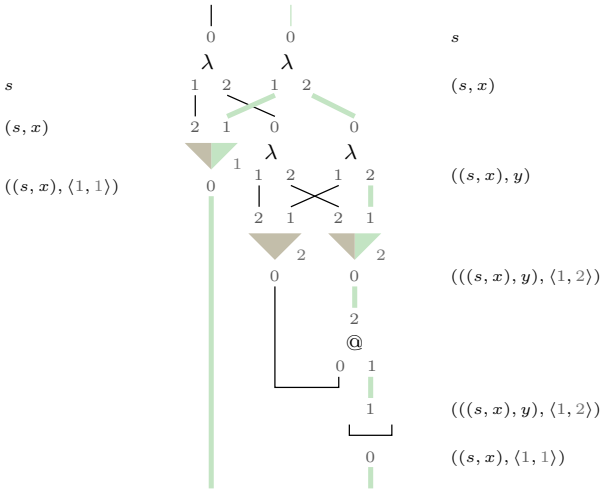
Als de eerste waaier in een paar op een abstractor inwerkt, verandert het uitgebreide label van die waaier terwijl het uitgebreide label van de waaier waarmee die waaier een paar

vormt hetzelfde blijft. Waaiers die elkaar voor wat betreft de stapel in evenwicht houden hebben dus niet altijd hetzelfde uitgebreide label.

Alle namen op één stapel

Wanneer je namen van de stapel moet halen, zou je naar het begin van de tak moeten zoeken. Omdat je wanneer je leest alleen de grens en niet de hele tak ziet, maar wel de stapel, bewaren we de namen van variabelen en die van de poorten op één stapel. Bij een grens halen we dan de poorten weg die op de eerste variabele in de stapel liggen.

Maar wanneer je alleen de naam van een poort op de stapel zou bewaren, weet je bij een grens nog steeds niet genoeg. Want alleen als je de naam bij een waaier die een 1 in haar label heeft op de stapel hebt gelegd, moet die er bij de grens vanaf. We bewaren daarom de uitbreiding van het label naast de naam van de poort.



(195)

Zoeken we, wanneer we de grens oversteken, naar namen van poorten, dan komen we de naam van de poort die we bij de waaier op de stapel hebben gelegd tegen. Aangezien er echter een 2 naast die naam ligt, laten we de naam gewoon op de stapel liggen.

Bij de volgende grens zoeken we ook weer naar namen waar een 1 naast ligt. Omdat we bij die grens de naam die we bij de waaier op de stapel leggen weer van de stapel moeten halen, moeten we bij de grens die bij de tweede abstractor hoort de aanduidingen bij namen die we nog wel zien met één verlagen.

De betekenis van waaiers met een uitgebreid label

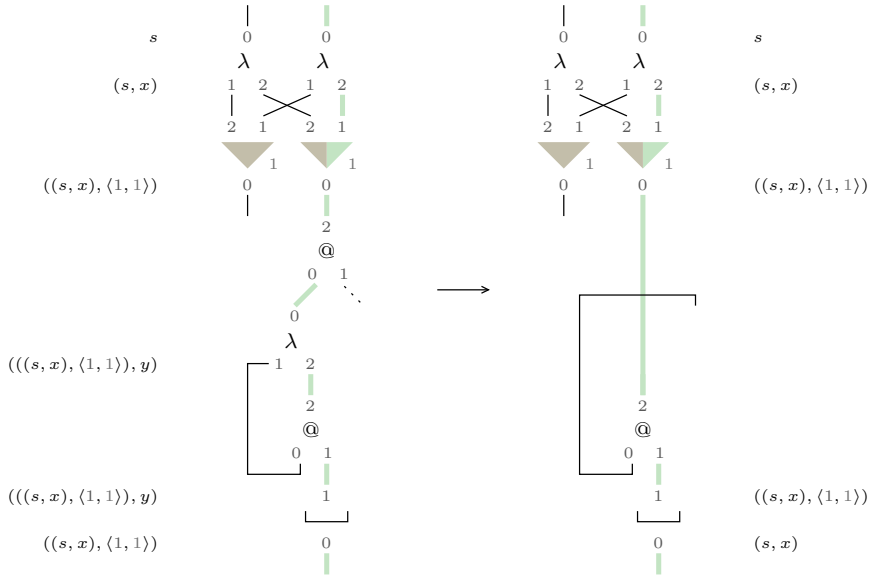
Wanneer we een graaf met enkelvoudige naamloze binding interpreteren, leggen we bij een abstractor de naam van een verse variabele op de stapel. Aangezien we bij een waaier met een uitgebreid label ook een naam op de stapel leggen, of die er, zoals bij een grens, weer vanaf halen, breiden de waaier, de abstractor en de grens de praktijk waarin we met λ-expressies rekenen op een vergelijkbare manier uit.

Zouden we de poorten die bij waaiers horen niet op dezelfde manier als de namen van variabelen met een stapel onthouden, dan moeten we de praktijk verder uitbreiden. Dit maakt het moeilijker grafen een betekenis te geven die binnen de praktijk ligt.

Het effect van een waaier met een uitgebreid label hoeft niet te veranderen als de abstractor die bij de naam hoort waar de waaier naar wijst uit de graaf verdwijnt. Als we er tenminste voor zorgen dat er in plaats van de naam van de variabele naam iets anders op de stapel komt te liggen, iets dat ons aan de variabele herinnert.

Geen abstractor, wel grenzen

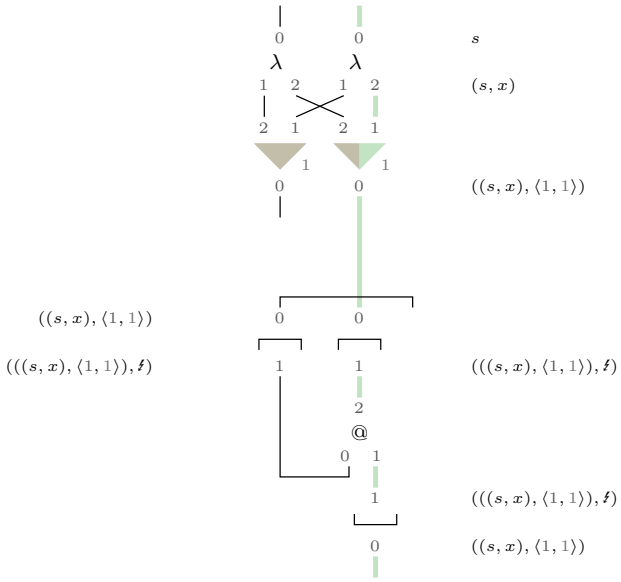
Wanneer we grafen met een stapel lezen, moeten de waaiers in de gebogen takken elkaar in evenwicht houden, en bij een grens moeten we de namen van de poorten van de stapel halen. Maar door de interactie van een applicator en een abstractor missen we de naam van de variabele die bij die abstractor hoorde, en daardoor halen we mogelijk te veel namen van poorten van de stapel.



(196)

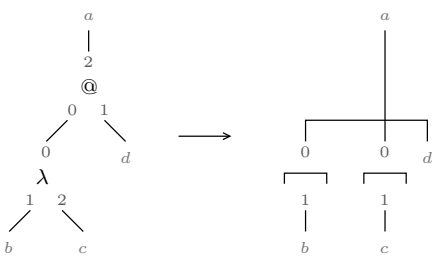
Wanneer we na de interactie aan het begin van de gebogen tak toch iets op de stapel zouden leggen, iets wat bij het zoeken in de stapel de rol van de variabele kan spelen, dan werken we de stapel op dezelfde manier bij, en bevat de stapel aan de andere kant van de $(1,0)$-grens vóór en na de interactie hetzelfde.

Omdat de context alleen door het lezen van een knoop kan veranderen, laten we de interactie de abstractor vervangen door een knoop die het gewenste effect heeft. Aangezien de $(1,0)$-grens het einde het bereik aangeeft, en die grens zelf niet verdwijnt, laten we de interactie in de tak waar we door de abstractor liepen een $(0,1)$-grens toevoegen. Wanneer we de $(0,1)$-grens passeren, leggen we een $\lightning$ op de stapel. We spreken af dat als we in de stapel naar een naam zoeken, de $\lightning$ de rol van de variabele speelt.



(197)

Omdat de interactie de tak die naar de $(1,0)$-grens leidt niet verandert, gaan we na die grens met dezelfde stapel verder als vóór de interactie. Voegt de interactie aan het einde van de gebogen tak bovendien een $(1,0)$-grens toe, dan halen we daar, omdat de interactie de knopen in de gebogen tak niet verandert, de $\lightning$ weer van de stapel.



(198)

Ook de gebogen tak verandert niet, en daarom beginnen we vóór en na de interactie in dezelfde context aan de rechter tak van de applicator. Als je niet terugkomt in het fragment dat verandert, dan komt de interactie met één substitutie overeen. De interactie zet in ieder geval een optimale stap.

Als de abstractors en grenzen elkaar vóór de interactie (198) in evenwicht houden, dan doen ze dat na de interactie ook. Maar wel op een andere manier: de eerste grens in het paar hoort bij de grens in een aftakking, echter ook bij de tweede grens in het paar.

Omdat abstractors en grenzen elkaar niet meer op dezelfde manier in evenwicht houden als in een e-graaf veranderen we de oorspronkelijke definitie (150) van het evenwicht.

Definitie: **grens-gebalanceerdheid** (199)

Stelt $\triangle$ een waaier met of zonder uitgebreid label, $-$ een grens, x een variabele, en ϵ de lege expressie voor, en kun je met de regels

$$\begin{aligned} e_0 &= -_{(0,1)}e_1-_{(1,0)} \mid \lambda_{(0,2)}e_1\lambda_{(1,0)} \\ e_1 &= @e_1 \mid \triangle e_1 \mid \lambda x\, e_0 \mid e_2e_1 \mid \epsilon \\ e_2 &= -_{(0,1)}e_1-_{(1,0)} \mid \lambda_{(0,2)}e_1-_{(1,0)} \end{aligned}$$

de opeenvolging van labels van knopen in een tak beschrijven, dan houden de abstractors en grenzen elkaar in die tak in evenwicht. □

We breiden de oorspronkelijke definitie uit door de eerste clausule niet alleen de tak tussen de 2-poort en de 1-poort van één en dezelfde abstractor te laten beschrijven. De tak kan ook beginnen in een $(0,1)$-grens en eindigen in een $(1,0)$-grens.

Omdat er de na de $(1,0)$-grens die de interactie (198) toevoegt een gebogen tak of een taak met eenzelfde paar grenzen kan volgen, kan het ene evenwicht van abstractors en grenzen het andere opvolgen. Vanwege de tweede clausule kan de definitie een dergelijk evenwicht beschrijven.

De oorspronkelijke definitie beperkt waaiers, want aan het einde van de gebogen tak kan in een e-graaf alleen nog een andere waaier volgen. Omdat de interactie van een applicator en een abstractor het einde van de gebogen tak met een andere knoop verbindt, vervalt deze beperking, en laten we de bijbehorende clausule hier achterwege.

Vanwege de interactie die de abstractor door een paar grenzen vervangt, houden een $(0,1)$-grens en een $(1,0)$-grens die een aftakking aangeeft elkaar in evenwicht. Hierdoor verandert de laatste clausule in de definitie.

Naast het feit dat je na een 1-waaier in een aftakking terecht kunt komen, en waaiers dus geen deel uit kunnen maken van het evenwicht van abstractors en grenzen, kan een waaier

ook over een $(0, 1)$-grens springen. Ook daarom kunnen waaiers geen deel uit maken van het evenwicht van abstractors en grenzen.

Omdat de interactie het evenwicht van abstractors en grenzen niet mag verstoren, laten we de grens-gebalanceerdheid (199) van takken deel uit maken van de o-invariant.

Over de betekenis van grenzen

Vanwege lokale binding hoef je in grafen met waaiers en grenzen geen rekening te houden met namen van variabelen. Ook in expressies met De Bruijn indices zie je de variabelen niet, maar je moet wel op de indices letten.

Bij reductie van een expressie daalt een index in de abstractie die niet verder wijst dan de abstractor die verdwijnt met één. Verder moeten we een index in het argument die verder wijst dan de applicator die verdwijnt de afstand van de index en de abstractor die verdwijnt optellen. Zo verandert

$$\lambda.(\lambda.\lambda.2(\lambda.20))\lambda.10$$

niet in

$$\lambda.\lambda.2(\lambda.(\lambda.10)0)$$

maar in

$$\lambda.\lambda.1(\lambda.(\lambda.30)0)$$

Aangezien de 0 in het argument niet verder wijst dan een abstractor in het argument, hoeven we die index niet te veranderen. Maar de 1 wijst wel verder, namelijk naar de eerste abstractor. Omdat er twee abstractors tussen de abstractor en de index die verdwijnt zitten, moeten we 2 bij die index optellen.

De verandering van een index kun je expliciet weergeven. Bird en Paterson [3, p.87-88] doen dit door voor iedere extra abstractor een bewerking die één bij de index optelt aan die index te laten voorafgaan. Met de opvolgerfunctie s geeft

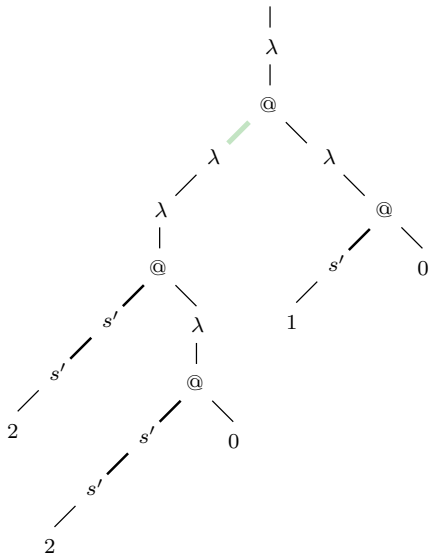
$$\lambda.\lambda.1(\lambda.(\lambda.ss10)0)$$

de expressie na de reductie weer. Zie je de bewerking ruim, dan kun je haar in plaats van op bepaalde indices, ook op het hele argument toepassen. Met s' de bewerking die één optelt bij indices die verder wijzen krijg je

$$\lambda.\lambda.1(\lambda.s's'(\lambda.10)0)$$

We zouden de bewerking s' ook onafhankelijk van de reductie kunnen introduceren: zodra we weten dat we in een tak de variabele die bij een abstractor hoort niet meer lezen, kunnen we vanwege die abstractor een s' toevoegen. De bewerkingen die op die manier

in een tak verschijnen passen we uiteindelijk toe op de index die zich aan het einde van de tak bevindt.



(200)

Als we de bewerking vóór de reductie aan de expressie toevoegen, gaat er niet alleen aan de indices in de abstractie een s' vooraf. Omdat aan de index 1 in het argument een abstractor voorafgaat die niet bij die index hoort, volgt er een s' in de tak. Zo lezen we

$$\lambda.\lambda.0(\lambda.s's'(\lambda.s'10)0)$$

in de syntaxboom van de expressie.

Wanneer je een expressie interpreteert door bij iedere abstractor een verse variabele op een stapel te leggen, wijst iedere index naar een variabele. Aangezien de reductie het argument verplaatst, verschijnen er vanuit het perspectief van de indices in dat argument extra variabelen op de stapel. De s' zou die variabelen daar weer vanaf kunnen halen. Als ze dat doet, speelt die bewerking dezelfde rol als een grens die een aftakking aangeeft.

De λ-rekenkunde bevat geen bewerking die een rol speelt die we met die van een grens of een s' kunnen vergelijken. De uitbreiding die Van Oostrom en Hendriks voorstellen [8] voegt echter een bewerking λ aan de rekenkunde toe die het einde van het bereik van een abstractor aangeeft.

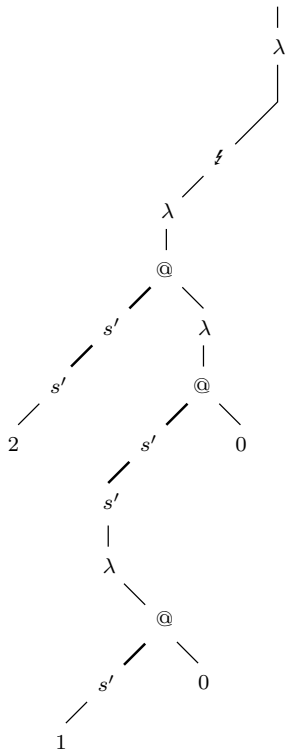
Als de λ's in een boom het bereik van de voorafgaande abstractor sluiten vóórdat ze het bereik van de abstractor sluiten die daar weer aan voorafgaat, omvat het bereik van de ene abstractor dat van de andere, en heeft een λ hetzelfde effect als een s' of een grens. De voorwaarde die we aan λ-expressies stellen komt overeen met de gebalanceerdheid (199) van abstractors en grenzen.

Net als enkelvoudige naamloze binding (170) houdt het met s' of λ sluiten van het bereik van een abstractor tegen dat de reductie van een expressie de binding van de variabelen verandert.

Omdat we een grens die het begin van een aftakking aangeeft op de bewerking s' en ook op λ kunnen betrekken, lukt het ons ook een dergelijke grens een betekenis te geven die op het vlak van de λ-rekenkunde ligt. We zien dat de grens de praktijk van het rekenen met λ-expressies niet verder uitbreidt dan λ dat doet.

De eerste grens in het paar (198) speelt echter een andere rol dan de grens die we op de bewerkingen s' en λ kunnen betrekken. Bij een $(1, 0)$-grens halen we namelijk, net als bij die bewerkingen, een naam van de stapel, terwijl we bij een $(0, 1)$-grens een $\lightning$ aan de stapel toevoegen.

We kunnen een bewerking aan de De Bruijn rekenkunde toevoegen die dezelfde rol speelt als een $(0, 1)$-grens. Zouden we namelijk in een expressies met indices wanneer we een $\lightning$ tegenkomen, deze niet lezen, maar wel meetellen bij het bepalen van een verwijzing naar abstractor, dan hoeven we de indices in de 0-tak van de eerste applicator (61) bij de reductie van de expressie niet aan te passen.



(201)

Zo wijst de 2 na de reductie naar dezelfde abstractor als ervoor. Omdat de eerste grens in

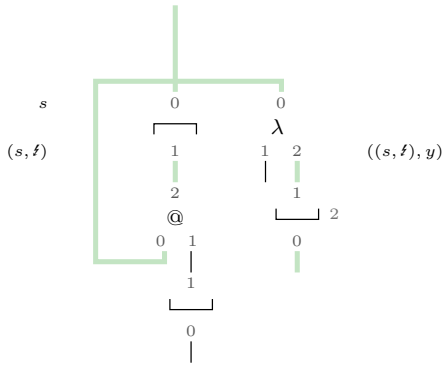
het paar (198) bewerkstelligt dat de namen van variabelen op de stapel niet opschuiven als we na de interactie de grens in een aftakking of de grens aan het einde van de gebogen tak oversteken, volgt dat we een $(0, 1)$-grens op de ↯ in een expressie met indices kunnen betrekken.

Omdat we bij de tweede grens die de interactie van een applicator en een abstractor (198) toevoegt net als bij een grens die een aftakking aangeeft iets van de stapel halen, kunnen we het effect van de $(1, 0)$-grens in het paar vergelijken met het effect van een grens die een abstractor in evenwicht (199) houdt.

Zowel de grenzen die een aftakking aangeven, als de grenzen die de interactie van een applicator en een abstractor toevoegen kunnen we dus een betekenis geven die op het vlak van de met de bewerking ↯ uitgebreide De Bruijn rekenkunde ligt. Wat grenzen betreft zouden interactie in grafen en de aangepaste De Bruijn rekenkunde de praktijk op dezelfde manier uitbreiden.

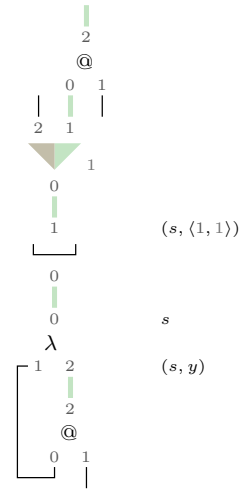
Grenzen verleggen, namen van poorten verbergen

Als er in de 0-tak van de applicator (198) na de grens een abstractor volgt, dan kan de applicator alleen op de abstractor inwerken als interactie eerst de grens achter de abstractor legt. Omdat de nieuwe grens niet bij de voorafgaande abstractor hoort, breidt de interactie die de grens verlegt het label van de grens uit.



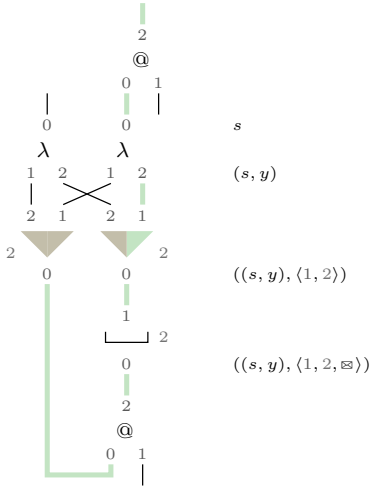
(202)

Door het label van de grens met 2 uit te breiden, geeft de interactie aan dat de grens niet bij de voorafgaande abstractor hoort, maar bij de $(0, 1)$-grens of abstractor die daar aan voorafgaat. Bij de 2-grens zouden we dus de ↯ van de stapel halen moeten halen. Maar de ↯ ligt onder de naam die we bij de abstractor op de stapel leggen.



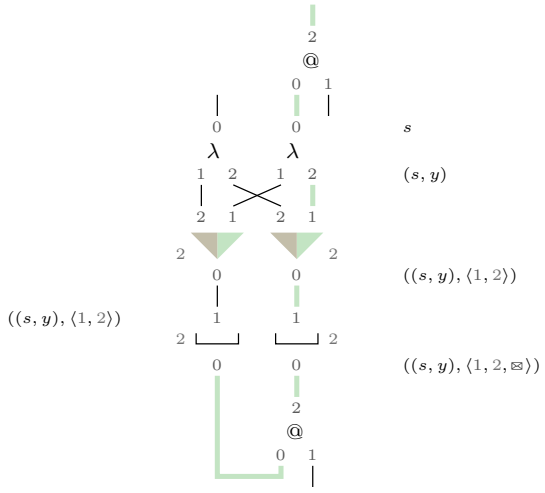
(203)

Zit er echter niet alleen een grens maar ook een waaier de interactie van een applicator en een abstractor in de weg, dan maakt dat de situatie nog lastiger. Na de interactie zouden we namelijk bij de 2-grens de naam van de poort die we bij de 2-waaier op de stapel leggen weer verwijderen. Dit terwijl we die naam bij de tweede waaier in het paar weer nodig hebben.



(204)

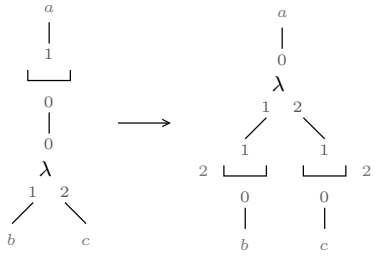
Het tegenstrijdige in de situatie verdwijnt als we de naam van de poort niet zouden zien wanneer we door de waaiers lopen. We maken de naam daarom onzichtbaar door er op de stapel een $\boxtimes$ naast te leggen, en af te spreken dat we, wanneer we door een waaier lopen en een richting moeten kiezen we niet naar namen met een $\boxtimes$ kijken.



(205)

Maar de naam die we bij de eerste waaier in het paar op de stapel leggen, en bij een grens onzichtbaar maken, hebben we bij de tweede waaier in het paar weer nodig. Daarom moeten we, nog vóór dat we in die waaier terechtkomen, eerst een andere knoop lezen, een knoop waarbij we de naam weer zichtbaar maken.

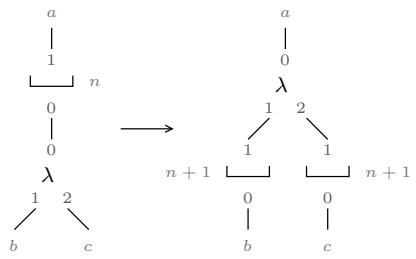
Omdat de knoop het effect dat de eerste grens in het paar (205) heeft op moet heffen, laten we de interactie een $(0, 1)$-grens toevoegen, en spreken we af dat wanneer we die grens oversteken de ✉ weer van de stapel halen.



(206)

Breidt de interactie het label van beide grenzen op dezelfde manier uit, dan zoeken we bij de eerste en tweede grens in het paar naar dezelfde naam. Vind je een naam, dan maak je deze weer zichtbaar door de ✉ weg te halen die naast de naam en de uitbreiding van het label van de waaier op de stapel ligt.

Ook als een n-grens de interactie van een applicator en een abstractor in de weg zit, legt interactie die grens achter de abstractor neer. Aangezien we na die interactie nog vóór we bij de grens komen eerst een variabele op de stapel leggen, zoeken we alleen naar dezelfde variabele of $\lightning$ als de interactie één bij de uitbreiding van het label optelt.



(207)

Omdat de interactie van een abstractor en een grens zonder uitgebreid label het label van de grenzen met 2 uitbreidt, kun je de regel toepassen op grenzen die een label met een natuurlijk getal $n, n \geq 2$ als uitbreiding hebben.

Werkt de grens aan het begin van de tak op een volgende abstractor in, dan krijgt die grens een ander label. Dit terwijl het label van de grens aan het einde van de gebogen tak niet verandert. Net als waaiers hebben grenzen in een paar niet altijd hetzelfde uitgebreide label.

Steken we een grens over, dan moeten we niet alleen namen van poorten, maar ook de naam van de variabele of de ↯ die bij de grens hoort van de stapel halen. Maar ook de ↯ of de naam van de variabele die we bij de oorspronkelijke grens van de stapel zouden halen, maken we met een ✉ alleen onzichtbaar. Want bij het zoeken naar namen van poorten begrenzen de ↯ en de namen van variabelen de namen die we zoeken.

Alleen bij grenzen die vanwege de interactie van een grens en een abstractor in de graaf verschijnen, verbergen we namen van poorten, of maken we die juist weer zichtbaar. Bij de grenzen zonder een uitgebreid label zoeken we gewoon in de stapel tot we de eerste variabele of ↯ tegenkomen.

Aangezien de interactie de volgorde van de grens en de abstractor verandert, krijgt het evenwicht van abstractors en grenzen een andere vorm. Stelt $-$ een grens met of zonder uitgebreid label voor, dan verandert de interactie een evenwicht van de vorm

$$(\lambda \cdots -)(\lambda \cdots -)$$

in

$$(\lambda \cdots (\lambda -) \cdots -)$$

Omdat een grens die de interactie achter een abstractor legt daar de 1-waaiers, en de grens die de aftakking aangeeft voorafgaat, ligt er aan het begin van een aftakking vóór en na die interactie hetzelfde op de stapel. Als we alleen naar de eerste grens in het paar kijken, verstoort de interactie (206, 207) het evenwicht van abstractors en grenzen niet.

Maar het evenwicht biedt geen plaats aan de grens die zich aan het einde van de gebogen

tak bevindt, en daarom leggen we het evenwicht van grenzen met een uitgebreid label op dezelfde manier vast als het evenwicht van waaiers die een uitgebreid label hebben.

Definitie: **grens-gebalanceerdheid** (208)

Stelt ϵ de lege expressie, $\triangle$ een waaier met of zonder uitgebreid label, x een variabele, $-$ een grens zonder en $-u$ een grens met een uitgebreid label voor, en kun je met de regels

$$\begin{aligned} e_0 &= \epsilon \mid @e_0 \mid \triangle e_0 \mid \lambda e_0 \mid \lambda x\, e_0 \mid -e_0 \mid e_1 e_0 \\ e_1 &= -u_{(1,0)} e_0 - u_{(0,1)} \end{aligned}$$

de opeenvolging van labels van knopen in een tak beschrijven, dan houden de grenzen die een uitgebreid label hebben elkaar in die tak in evenwicht. □

Als de gebogen tak door interactie van een applicator en een abstractor verandert in een tak tussen twee grenzen zonder uitgebreid label, kan het ene evenwicht van grenzen met een uitgebreid label op een ander evenwicht volgen. De laatste term van de eerste clausule beschrijft dit.

Aangezien de interactie in een graaf het evenwicht van abstractors en grenzen (199) net als dat van grenzen met een uitgebreid label (208) niet mag verstoren, maken beide vormen van evenwicht deel uit van de o-invariant.

Hoe stapels er inmiddels uitzien

Omdat we niet alleen namen van poorten maar ook de namen van variabelen op de stapel bewaren, een variabele door een $\lightning$ vervangen, en aan de naam van een poort de uitbreiding van het label van een waaier en mogelijk ook een $\boxtimes$ toevoegen, breiden we de definitie (187) van de stapels uit.

Definitie: **stapels** (209)

Met V de verzameling verse variabelen en $n \in \mathbb{N}$ leggen

$$\begin{aligned} & & & & & & \epsilon \in S \\ x \in V & & & & & \to & x \in S \\ & & & & & & \lightning \in S \\ x \in V & & & & & \to & \langle x, \boxtimes \rangle \in S \\ & & & & & & \langle \lightning, \boxtimes \rangle \in S \\ x \in V & \wedge & s \in S & \wedge & s \neq \epsilon & \to & (s, x) \in S \\ x \in V & \wedge & s \in S & \wedge & s \neq \epsilon & \to & (s, \langle x, \boxtimes \rangle) \in S \end{aligned}$$

$$
\begin{array}{rcrcrcr}
 & & s \in S & \wedge & s \neq \epsilon & \rightarrow & (s, \lightning) \in S \\
 & & s \in S & \wedge & s \neq \epsilon & \rightarrow & (s, \langle \lightning, \bowtie \rangle) \in S \\
p \in \{1,2\} & \wedge & s \in S & \wedge & s \neq \epsilon & \rightarrow & (s, \langle p, n \rangle) \in S \\
p \in \{1,2\} & \wedge & s \in S & \wedge & s \neq \epsilon & \rightarrow & (s, \langle p, n, \bowtie \rangle) \in S
\end{array}
$$

de verzameling stapels S vast. □

Op stapels treffen we het volgende aan: de naam van een variabele, een $\lightning$ die de naam van een variabele vervangt, of de naam van een poort van een waaier. De naam van een poort gaat in ieder geval vergezeld van een index, de verwijzing naar de $\lightning$ of de abstractor die een waaier aanwijst. Als er een $\bowtie$ naast een naam of een $\lightning$ ligt, dan is deze naam of $\lightning$ beperkt zichtbaar.

Het effect van grenzen op stapels

Omdat we bij het passeren van grenzen, niet eenvoudig een naam van de stapel halen of erop leggen, leggen we vóór dat we een afbeelding vastleggen die expressies aan grafen toevoegt, eerst afbeeldingen vast die het effect van grenzen op de stapel weergeven. We beginnen met grenzen die geen uitgebreid label hebben.

Definitie: **het effect van een $(1,0)$-grens zonder uitgebreid label** (210)

Met $s_1, s_2 \in S$, $s_2 \neq \epsilon$, $x \in V$, $p \in \{1,2\}$ en $n \in \mathbb{N}$, leggen

$$
\begin{array}{rcrcl}
 & & s_1 = x & \rightarrow & \succ\!\!\!\circ(s_1) = \epsilon \\
 & & s_1 = \lightning & \rightarrow & \succ\!\!\!\circ(s_1) = \epsilon \\
 & & s_1 = (s_2, x) & \rightarrow & \succ\!\!\!\circ(s_1) = s_2 \\
 & & s_1 = (s_2, \lightning) & \rightarrow & \succ\!\!\!\circ(s_1) = s_2 \\
 & & s_1 = (s_2, \langle p, 1 \rangle) & \rightarrow & \succ\!\!\!\circ(s_1) = \succ\!\!\!\circ(s_2) \\
n > 1 & \wedge & s_1 = (s_2, \langle p, n \rangle) & \rightarrow & \succ\!\!\!\circ(s_1) = (\succ\!\!\!\circ(s_2), \langle p, n-1 \rangle) \\
n > 1 & \wedge & s_1 = (s_2, \langle p, n, \bowtie \rangle) & \rightarrow & \succ\!\!\!\circ(s_1) = (\succ\!\!\!\circ(s_2), \langle p, n-1, \bowtie \rangle)
\end{array}
$$

een afbeelding $\succ\!\!\!\circ : S \rightarrow S$ vast: $\succ\!\!\!\circ(s)$ is de stapel na een grens die we met een stapel s naderen. □

De eerste vier clausules in de definitie leggen vast dat we in de stapel niet verder zoeken dan de eerste $\lightning$ of variabele die we tegenkomen. De clausules geven aan dat die $\lightning$ of naam verdwijnt. Vanwege de vijfde clausule verdwijnen alleen die namen die we bij het doorlopen van een 1-waaier op de stapel leggen. De zesde en zevende clausule

zorgen ervoor dat na de grens de uitbreiding die naast een naam ligt overeenkomt met de uitbreiding van het label van de waaier waarbij we de naam weer van de stapel moeten halen.

Naast een stapel heeft de afbeelding die het effect van grenzen met een uitgebreid label beschrijft de uitbreiding van het label van die grens als parameter.

Definitie: **het effect van grenzen met een uitgebreid label** (211)

Met $x \in V$, $s_1, s_2 \in S$, $s_2 \neq \epsilon$, $p \in \{1, 2\}$ en $n_1, n_2 \in \mathbb{N}$, leggen

$$
\begin{array}{lcll}
n_1 > n_2 & \wedge & s_1 = (s_2, \langle p, n_2 \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1, s_2 \rangle), \langle p, n_2 \rangle) \\
n_1 > n_2 & \wedge & s_1 = (s_2, \langle p, n_2, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1, s_2 \rangle), \langle p, n_2, \boxtimes \rangle) \\
 & & & \\
n_1 < n_2 & \wedge & s_1 = (s_2, \langle p, n_2 \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1, s_2 \rangle), \langle p, n_2 - 1 \rangle) \\
n_1 < n_2 & \wedge & s_1 = (s_2, \langle p, n_2, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1, s_2 \rangle), \langle p, n_2 - 1, \boxtimes \rangle) \\
 & & & \\
n_1 > 0 & \wedge & s_1 = \langle s_2, x \rangle & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1 - 1, s_2 \rangle), x) \\
n_1 > 0 & \wedge & s_1 = (s_2, \langle x, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1 - 1, s_2 \rangle), \langle x, \boxtimes \rangle) \\
 & & & \\
n_1 > 0 & \wedge & s_1 = (s_2, \lightning) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1 - 1, s_2 \rangle), \lightning) \\
n_1 > 0 & \wedge & s_1 = (s_2, \langle \lightning, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1 - 1, s_2 \rangle), \langle \lightning, \boxtimes \rangle) \\
 & & & \\
n_1 = n_2 & \wedge & s_1 = (s_2, \langle p, n_2 \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1, s_2 \rangle), \langle p, n_2, \boxtimes \rangle) \\
n_1 = n_2 & \wedge & s_1 = (s_2, \langle p, n_2, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (\triangle_{\boxtimes}(\langle n_1, s_2 \rangle), \langle p, n_2 \rangle) \\
 & & & \\
n_1 = 0 & \wedge & s_1 = (s_2, x) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (s_2, \langle x, \boxtimes \rangle) \\
n_1 = 0 & \wedge & s_1 = (s_2, \langle x, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (s_2, x) \\
 & & & \\
n_1 = 0 & \wedge & s_1 = (s_2, \lightning) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (s_2, \langle \lightning, \boxtimes \rangle) \\
n_1 = 0 & \wedge & s_1 = (s_2, \langle \lightning, \boxtimes \rangle) & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = (s_2, \lightning) \\
 & & & \\
n_1 = 0 & \wedge & s_1 = x & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = \langle x, \boxtimes \rangle \\
n_1 = 0 & \wedge & s_1 = \langle x, \boxtimes \rangle & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = x \\
 & & & \\
n_1 = 0 & \wedge & s_1 = \lightning & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = \langle \lightning, \boxtimes \rangle \\
n_1 = 0 & \wedge & s_1 = \langle \lightning, \boxtimes \rangle & \rightarrow \quad \triangle_{\boxtimes}(\langle n_1, s_1 \rangle) = \lightning
\end{array}
$$

een afbeelding $\triangle_{\bowtie} : \mathbb{N} \times S \to S$ vast: $\triangle_{\bowtie}(\langle n, s\rangle)$ is de stapel na een n-grens die we met stapel s naderen. □

De eerste vier clausules geven aan dat de zichtbaarheid van een naam niet verandert als het label van de waaier waar de naam bijhoort een andere uitbreiding heeft als het label van de grens. Als $n_1 < n_2$ hoort de naam bij een waaier die oorspronkelijk aan de abstractor waar de grens bijhoort voorafgaat. Op dezelfde manier als bij een grens zonder uitgebreid label (210) daalt de uitbreiding in dat geval met één.

Het derde en het vierde paar clausules leggen vast dat als we een $\lightning$ of de naam van een variabele tegenkomen die niet bij de grens hoort, en we verder gaan met zoeken. De zichtbaarheid van de naam of $\lightning$ verandert niet. Omdat we naar namen zoeken die bij een volgende naam of $\lightning$ horen, daalt de waarde van de eerste parameter.

De clausules in het vijfde paar beschrijven wat er gebeurt als we een naam van een poort aantreffen waarvan de zichtbaarheid moet veranderen. Omdat er nog een naam kan volgen waarmee hetzelfde moet gebeuren, blijft de eerste parameter hetzelfde.

Naast de zichtbaarheid van namen van poorten veranderen ook de zichtbaarheid van de $\lightning$ of de naam van de variabele die bij de grens hoort. De clausules in het zesde en zevende paar geven dit weer. Ze geven ook aan dat we niet verder hoeven te zoeken.

Het achtste en negende paar hebben we nodig omdat in plaats van de stapel zoals we die in de clausules van het voorafgaande paar zien, de stapel ook uit één enkele variabele of $\lightning$ kan bestaan.

Omdat de eerste parameter in de afbeelding, niet groter mag zijn dan het aantal namen van variabelen of $\lightning$ op de stapel, kunnen we de afbeelding $\triangle_{\bowtie}$ niet als totaal opvatten.

We gebruiken $\rightarrowtail$ en $\triangle_{\bowtie}$ in de definitie van de afbeelding van grafen op expressies. Omdat een $(0, 1)$-grens zonder uitgebreid label een beperkt effect op de context heeft, leggen we dit effect niet apart vast. Net als het effect van de andere knopen zien we het in onderdeel 6 in de definitie van de afbeelding terug.

De betekenis van grenzen met een uitgebreid label

We kunnen een expressie in de De Bruijn rekenkunde als gewone λ-expressie interpreteren, door bij iedere abstractor die we tegenkomen de naam van een verse variabele op een stapel te leggen. Wanneer we die rekenkunde met de bewerkingen $\lightning$ en s' uitbreiden, zien we bovendien dat die hetzelfde effect hebben als een grens zonder uitgebreid label.

De bewerking s' die aan een index voorafgaat heeft altijd betrekking een voorafgaande abstractor: de eerste s' sluit het bereik van de voorafgaande abstractor, de volgende s' sluit het bereik van de abstractor die daar weer aan voorafgaat. Een grens met een uitgebreid

label hoort ook bij een voorafgaande abstractor, maar niet noodzakelijk de eerstvolgende.

Omdat we de bewerking λ op een variabele toepassen kunnen we met die bewerking wel naar een abstractor wijzen die verder weg ligt. Zo sluit λx in

$$\lambda x.\lambda y.x\lambda xy$$

het bereik van de abstractor die aan λy voorafgaat. Als iedere λ in de expressie het bereik van een abstractor sluit, hebben te maken met wat Van Oostrom en Hendriks [8, definitie 2] een bereik-gebalanceerde expressie noemen.

Omdat we de variabele die bij een abstractor hoort niet onmiddellijk van de stapel halen zodra we een grens met een uitgebreid label passeren die naar een voorafgaande abstractor wijst, kunnen we het effect van een dergelijke grens vergelijken met het effect van een λ in een bereik-gebalanceerde expressie.

Hoewel we een grens met een λ kunnen vergelijken, verschillen beide ook van elkaar: de λ sluit een bereik definitief, terwijl we in een graaf bij een grens met een uitgebreid label een naam vanwege gebogen takken alleen maar onzichtbaar kunnen maken. Dit betekent dat je een grens met een uitgebreid label niet als een bewerking kunt zien die het einde van het bereik van een abstractor aangeeft.

Omdat we een grens met en zonder een uitgebreid op een λ kunnen betrekken, en het effect van een waaier met een uitgebreid label op dat van abstractor of grens lijkt, kunnen we o-grafen een betekenis geven die op het vlak van λ-expressies ligt: o-interactie breidt de praktijk van het rekenen met λ-expressies niet verder uit dan s' of λ dat doen.

Waar interactie zich aan moet houden

De interactie die ervoor zorgt dat een applicator op een abstractor in kan werken, voegt waaiers en grenzen aan de gebogen takken van abstractors toe. Omdat we bij het lezen van een graaf een stapel gebruiken om namen van poorten en variabelen te onthouden, mag de interactie het evenwicht van waaiers en het evenwicht van grenzen niet verstoren.

Definitie: **o-invariant** (212)

De gebalanceerdheid van waaiers (186) abstractors en grenzen (199) en grenzen met een uitgebreid label (208) in een gebogen tak, of een tak tussen de grenzen die de interactie van een applicator en een abstractor aan de graaf toevoegt. □

Omdat we belangstelling hebben voor de variabele die we aan het einde van een tak lezen, en vanwege de afspraak (182) bij de tweede waaier in een paar de juiste poort moeten kiezen, mag de interactie in het bijzonder het evenwicht in de gebogen takken van een o-graaf niet verstoren.

We verwachten van de interactie in een graaf dat ze iedere applicator die een abstractor in haar 0-tak heeft met die abstractor verbindt. Zodra we alle regels hebben vastgelegd die de interactie daarvoor nodig heeft, kunnen we laten zien dat de interactie het evenwicht in de takken in de graaf niet verstoort.

Aangezien de interactie die we vastleggen zich op het zetten van optimale stappen richt, spreken we over o-interactie, en over een o-invariant. Als de interactie zich aan de invariant houdt, dan kunnen we aan de hand van het effect (211) dat de verschillende soorten knopen op de stapel hebben, laten zien dat wanneer we een graaf in een context (209) lezen, we inderdaad de expressie krijgen die we verwachten.

5 Alle nodige o-regels

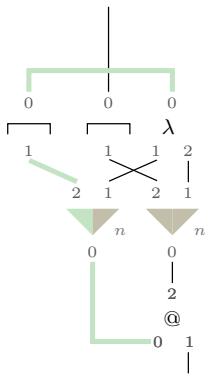
De interactie die ruimte maakt voor optimale stappen heeft met beperkingen te maken: ze mag de expressie die we in de graaf lezen niet veranderen, ze moet het evenwicht van waaiers en het evenwicht van abstractor en grenzen in stand houden, en ze mag de lokale binding van instanties van een variabele niet verbreken.

We leggen de regels die zich richten op het zetten van optimale stappen vast. Dit doen we door na te gaan of een tweetal knopen de interactie van een applicator en een abstractor in de weg kan zitten. Vinden we een dergelijke situatie, dan hoort daar een o-regel bij, een regel waarmee de interactie de knoop dichter bij de applicator of abstractor brengt, of nog beter: haar uit de tak haalt. Wanneer we verwachten dat een bepaalde situatie zich niet voor kan doen, dan laten we dat ook zien.

De interactie van een applicator en een waaier met een uitgebreid label

Na de waaier en de abstractor (191, 192) en de grens en de abstractor (206, 207) kijken we naar de interactie tussen een applicator en een waaier die een uitgebreid label heeft.

Bij de interactie van een waaier en een abstractor verschijnen er twee abstractors en twee waaiers in de graaf. Verdwijnt één van die abstractors door interactie met een applicator, en lezen we in de gebogen tak een applicator, dan zit één van de waaiers de interactie van die applicator en de overgebleven abstractor in de weg.

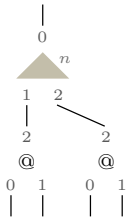


(213)

Omdat we in de 0-tak van de applicator als eerste symbool een abstractor lezen, hebben we met een reduceerbare expressie te maken. Maar de applicator kan alleen dan op de abstractor inwerken als de waaier in plaats van erop te volgen, aan de applicator vooraf zou gaan.

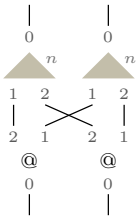
Als we eerst de waaier en daarna de applicator tegen willen komen, dan moeten we het

fragment met de applicator en de waaier in ieder geval vervangen door een partiële graaf (53) waarin niet één maar twee applicators zitten. Want zowel de 1-tak als de 2-tak van de waaier moet dan met een dergelijke knoop beginnen.



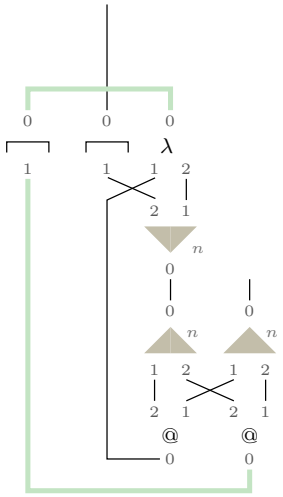
(214)

Het oorspronkelijke fragment zit met de 1-tak van de applicator aan de rest van de graaf vast. Omdat er in het nieuwe fragment niet één maar twee applicators zitten, hebben we een waaier nodig die de twee 1-takken van die knopen samenvoegt. De 0-tak van die waaier verbindt het fragment dan weer met de rest van de graaf.



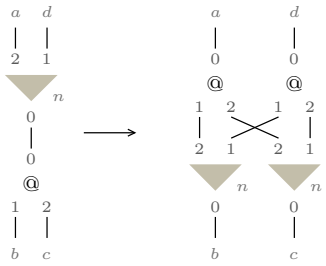
(215)

Omdat er twee applicators in het nieuwe fragment zitten, bevat dit fragment niet alleen twee 1-takken, maar ook twee 0-takken. Dit komt goed uit, want in het oorspronkelijke fragment (213) splitst de 0-tak van de applicator zich na de waaier. Het nieuwe fragment (215) kan daarmee het oorspronkelijke vervangen.



(216)

Vanwege het feit dat het voor wat betreft de context waarin we lezen geen verschil maakt of we vóór of na de applicator een naam van de stapel halen, en alleen de applicator deel uit maakt van de expressie, verwachten we niet dat de verwisseling de expressie die we lezen verandert. Regels van de volgende vorm lijken daarom geschikt.



(217)

De interactie (216) verstoort het evenwicht (186) niet: naast de waaier komen we alleen een applicator tegen, we doorlopen de waaier in dezelfde richting, en het uitgebreide label verandert niet.

Situaties met een applicator en een waaier zonder een uitgebreid label

Omdat zowel een applicator als een waaier zonder uitgebreid label met haar 0-poort naar het einde van een gebogen tak wijst, verwachten we niet dat we een regel voor de interactie van deze knopen nodig hebben.

Door de invariant uit te breiden laten we zien dat een waaier zonder uitgebreid label niet op een applicator in kan werken. Wanneer alle regels vastliggen bewijzen we dat de interactie zich daadwerkelijk aan de invariant houdt. Vooralsnog veronderstellen we dit alleen.

Definitie: **o-invariant** (218)

De oorspronkelijke invariant (212) en het feit dat een waaier met de 0-poort naar het einde van de tak, en een applicator met de 2-poort naar de wortel wijst. □

Stelling: **waaiers en applicators** (219)

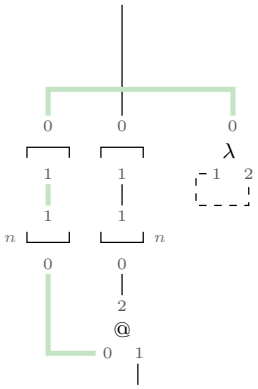
Een waaier zonder uitgebreid label zit in geen enkele o-graaf met haar 0-poort aan de 0-poort van een applicator vast.

Bewijs: veronderstel dat een waaier met de 0-poort aan de 0-poort van een applicator vastzit. Vanwege de invariant wijst die waaier dan naar de wortel. Maar dit spreekt de invariant tegen, want die zegt dat een waaier met haar 0-poort naar het einde van een gebogen tak, en dus niet naar de wortel wijst. □

Vanwege de stelling (219) volgt dat we geen regel nodig kunnen hebben voor de interactie van een applicator en een waaier zonder uitgebreid label.

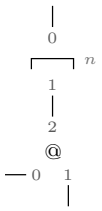
De interactie van een applicator en een grens met een uitgebreid label

Als een abstractor die een applicator en een paar grenzen in haar gebogen tak heeft op een applicator inwerkt, en de 1-tak van die applicator met een abstractor begint, dan krijgen we een situatie waarin we een expressie met een applicatie en een abstractie lezen. Omdat er echter grenzen tussen de applicator en de abstractor zitten, zou de interactie tussen deze knopen uitblijven.



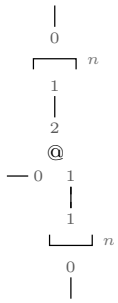
(220)

Als we in plaats van in de 0-tak van de applicator de grens nog vóór de applicator tegen zouden komen, zit er één knoop minder tussen de applicator en de abstractor.



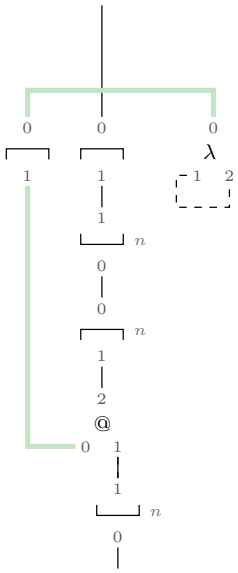
(221)

Omdat we in het nieuwe fragment met dezelfde context aan de 1-tak van de applicator willen beginnen, voegen we aan het begin van die tak een grens toe die het effect van de n-grens die onmiddellijk aan de applicator voorafgaat teniet doet.



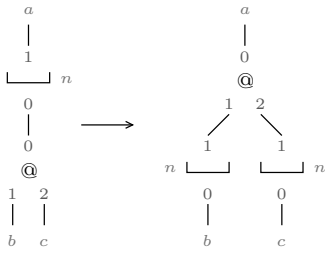
(222)

Het nieuwe fragment past in de graaf, en omdat we alleen de applicator en niet de grenzen in de expressie terugzien, en de context op dezelfde manier verandert als in het oude fragment, verandert de expressie die we lezen niet.



(223)

Regels van de volgende vorm lijken daarom geschikt voor de interactie van een applicator en een grens met een uitgebreid label.

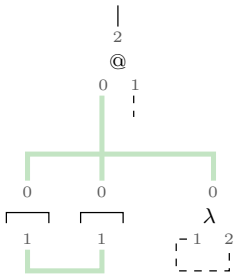


(224)

Omdat een applicator geen invloed heeft op het evenwicht (208) in de takken, en we vóór en na de interactie in het fragment in beide takken een grens met een uitgebreid label in de dezelfde richting oversteken, verstoort de interactie het evenwicht niet.

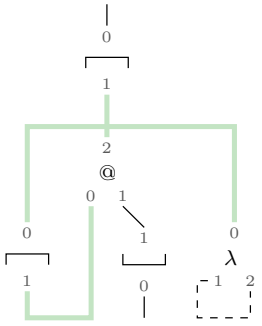
De interactie van een applicator en een grens die geen uitgebreid label heeft

Als in de tak tussen een applicator en een abstractor met de interactie van een andere applicator en een abstractor een paar grenzen zonder uitgebreid label verschijnt, lezen we een expressie met een abstractie in een applicatie terwijl in de 0-tak van de applicator een grens zonder een uitgebreid label aan de abstractor voorafgaat.



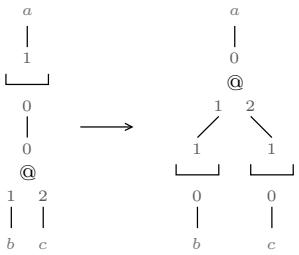
(225)

Zouden we eerst de grens en dan de applicator tegenkomen, dan bevindt er zich één knoop minder tussen de applicator en de abstractor. Net als in het geval van de interactie van een applicator en een grens met een uitgebreid label heft een grens in de 1-tak van de applicator het effect van de grens die onmiddellijk aan de applicator voorafgaat op.



(226)

De volgende regel lijkt daarom geschikt voor de interactie van een applicator en een grens zonder uitgebreid label.

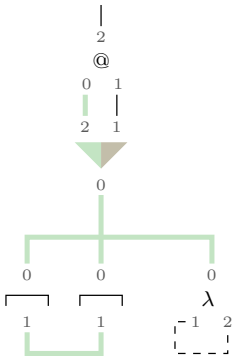


(227)

Op dezelfde manier als de regel (224) voor de interactie van een applicator en een grens met een uitgebreid label verstoort de interactie van een applicator en een grens zonder uitgebreid label het evenwicht in de takken niet.

De interactie van een waaier en een grens, beide zonder een uitgebreid label

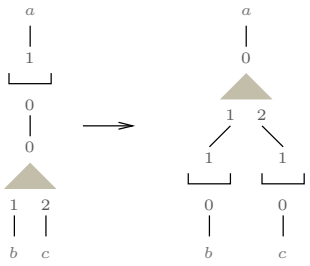
Als er een waaier zonder uitgebreid label aan de eerste grens (225) voorafgaat, lezen we ook weer een expressie met een abstractie in een applicatie. In dit geval zitten er niet alleen twee grenzen, maar ook nog een waaier zonder uitgebreid label in de 0-tak van de applicator.



(228)

Zouden we eerst de grens en daarna de waaier tegenkomen, dan kan de grens op de applicator inwerken. Verder kan de tweede grens en daarna de waaier op de abstractor inwerken. De interactie maakt de afstand tussen de applicator en abstractor echter niet kleiner, en daarom hebben we nog meer regels nodig.

Omdat het niet uitmaakt of we eerst door de waaier lopen en daarna een $\lightning$ of een naam van de stapel halen, of dit in de omgekeerde volgorde doen, verandert de context vóór en na een verwisseling op dezelfde manier. We lezen ook dezelfde expressie, want een waaier of een grens zien we niet in de expressie terug. Regels van de volgende vorm lijken daarom geschikt.

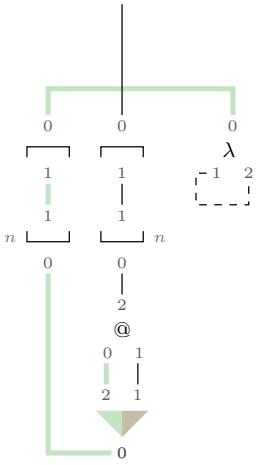


(229)

Aangezien een waaier die geen uitgebreid label heeft geen invloed kan hebben op het evenwicht van waaiers (186) en ook niet op dat van grenzen (199) houdt interactie die de regel volgt het evenwicht van waaiers en dat van grenzen in stand.

De interactie van een waaier zonder, en een grens met een uitgebreid label

We hebben ook een regel nodig voor de interactie van een waaier zonder en een grens met een uitgebreid label. Want de tak tussen de applicator en de abstractor (220) kan naast de grenzen ook een waaier bevatten die onmiddellijk op de applicator volgt.

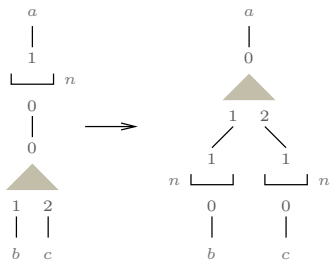


(230)

Zou de volgorde waarin we door de waaier en de grens lopen veranderen, en we eerst de grens en daarna de waaier tegenkomen, dan nadert de grens de applicator, en de waaier de abstractor. Dit kan de applicator en de abstractor uiteindelijk dichter bij elkaar brengen.

De waaier heeft geen effect op de context, en daardoor maakt de verwisseling geen verschil voor de manier waarop de context verandert. Omdat we bovendien de waaier en de grens niet in de expressie terugzien, lezen we vóór en na de interactie dezelfde expressie; een

regel van de volgende vorm lijkt daarom geschikt.

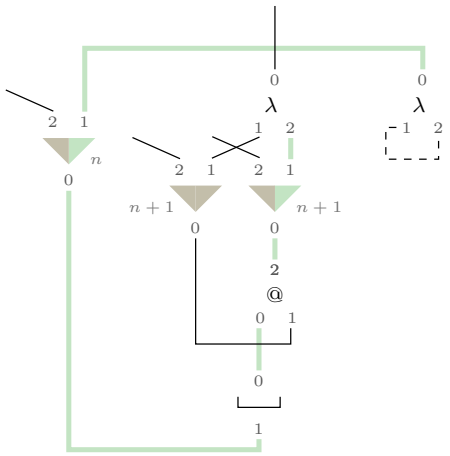


(231)

Omdat we vóór en na de interactie de grens met hetzelfde uitgebreide label in dezelfde richting oversteken, en een waaier die geen uitgebreid label heeft geen betrekking heeft op het evenwicht (186) van waaiers en ook niet op dat (208) van grenzen, bewaart interactie die de regel volgt het evenwicht in de takken.

De interactie van een waaier met en een grens zonder een uitgebreid label

Als één van de waaiers in een paar op een volgende abstractor inwerkt, dan vormt de eerste waaier in de gebogen tak van die abstractor nog steeds een paar met de waaier aan het einde van de gebogen tak van de voorafgaande abstractor. Vanwege het evenwicht van abstractors en grenzen (199) gaat er een grens aan die waaier vooraf.

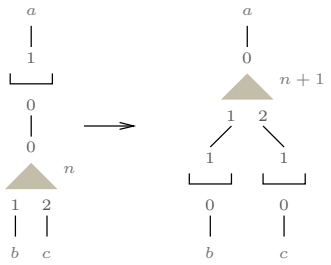


(232)

Zou de volgorde van de grens en de n-waaier veranderen, dan brengt dat die waaier dichter bij de applicator en de grens dichter bij de abstractor.

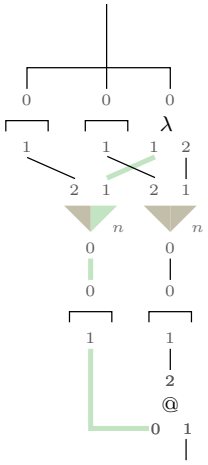
Verwisselen we de grens en de waaier, dan wijst de waaier, omdat we bij de grens iets

van de stapel halen, alleen nog naar dezelfde ↯ of naam van een variabele als we één bij de uitbreiding van het label optellen. Omdat we alleen knopen verwisselen die we toch niet in de expressie terugzien, verandert de expressie niet, en lijken regels van de volgende vorm geschikt.



(233)

Ook de eerste grens in een paar kan naar het begin van een gebogen tak wijzen. Een grens die dit doet maakt deel uit van het paar grenzen dat de interactie van een applicator en een abstractor aan de gebogen tak van die abstractor toevoegt.



(234)

Op dezelfde manier als in de vorige situatie volgt dat de regel (233) ook geschikt is indien de grens deel uitmaakt van een paar en naar het begin van de gebogen tak wijst.

Omdat waaiers die een paar vormen geen deel uitmaken van het evenwicht van grenzen, en het aantal waaiers in iedere tak hetzelfde blijft, houdt de interactie het evenwicht van waaiers (186) in stand. Omgekeerd volgt dat de interactie ook het evenwicht van grenzen (199) niet verstoort.

Situaties met twee waaiers die geen van beide een uitgebreid label hebben

Waaiers die geen van beide een uitgebreid label hebben, wijzen naar het einde van een gebogen tak. We verwachten daarom niet dat we een situatie kunnen vinden waarin deze knopen op elkaar in moeten werken.

Stelling: **waaiers zonder uitgebreide labels** (235)

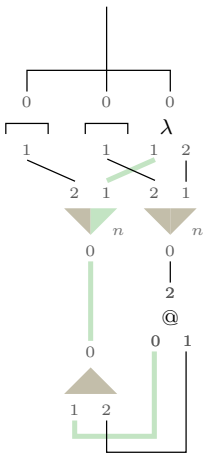
Een waaier die geen uitgebreid label heeft, zit in geen enkele o-graaf met haar 0-poort aan de 0-poort van een waaier vast die ook geen uitgebreid label heeft.

Bewijs: houdt de interactie in de graaf zich aan de invariant, dan wijzen waaiers zonder uitgebreid label naar het einde van een gebogen tak. Ze kunnen daardoor niet op elkaar inwerken. □

Als de interactie zich inderdaad aan de invariant houdt, volgt met het bewijs van de stelling dat we geen regel nodig hebben voor de interactie van waaiers zonder uitgebreid label.

De interactie van één waaier met en één waaier zonder uitgebreid label

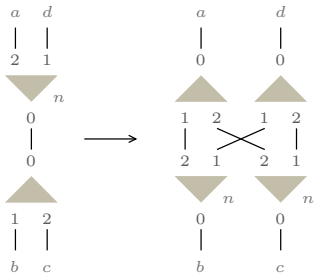
Als een abstractor meer dan één gebogen tak heeft, voegen één of meerdere waaiers deze takken samen zodat ze allemaal in de 1-poort van die abstractor eindigen. In een situatie waarin twee waaiers (213) de interactie van een applicator in de weg zitten, kan het paar dus een derde waaier omvatten.



(236)

Omdat een waaier die geen paar vormt met een andere waaier geen effect heeft op de context, en we bovendien zowel waaiers die geen paar vormen met een andere waaier als waaiers die dat wel doen niet terugzien in de expressie die we lezen, laten we de interactie

de volgorde waarin we de waaiers tegenkomen veranderen.

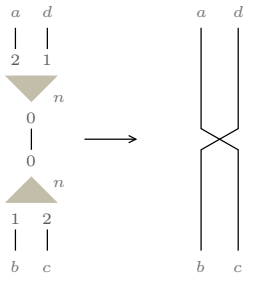


(237)

Wanneer interactie een regel volgt die deze vorm heeft, bewaart ze het evenwicht (186) van waaiers. Dit evenwicht heeft namelijk geen betrekking op een waaier zonder uitgebreid label, en vóór en na de interactie lopen we in dezelfde richting door een waaier met hetzelfde uitgebreide label.

De interactie van waaiers, beide met hetzelfde uitgebreide label

Wanneer de tak (216) deel uitmaakt van de 0-tak van een applicator, en er later in die tak nog een abstractor volgt, zou het verdwijnen van de waaiers die applicator en abstractor dichter bij elkaar brengen. Omdat de ene waaier het effect van de andere opheft, en we ze geen van beide in de expressie terugzien, lijkt een regel die ze opheft geschikt.



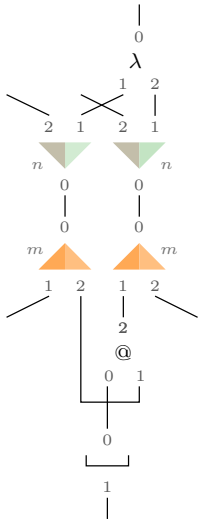
(238)

Na de interactie kruisen de takken elkaar. Dit komt omdat we de afspraak (182) over het lezen van waaiers volgen: in de tweede waaier van het paar kiezen we de poort waardoor we de eerste waaier binnengaan.

Omdat de waaiers die verdwijnen elkaar in evenwicht houden, verandert de interactie die de regel volgt niets aan het evenwicht in de takken (216) van de graaf.

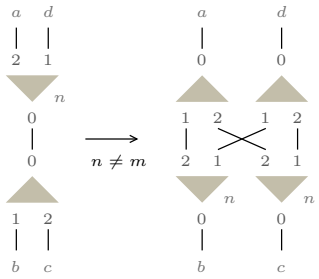
De interactie van waaiers met verschillende uitgebreide labels

Als de waaier in de 0-tak van de applicator (232) geen paar vormt met de eerste waaier in de gebogen tak van de abstractor, dan komt deze waaier na interactie met de grens en de applicator in de gebogen tak van de abstractor terecht. Omdat de waaier geen paar vormt met de waaier in de tak, hebben beide waaiers verschillende uitgebreide labels.



(239)

Volgt er na de grens een abstractor, dan verdwijnt de applicator. Volgt er nog een abstractor, en gaan er tenminste twee applicators aan het fragment vooraf, dan zitten de waaiers de interactie van een applicator en een abstractor in de weg. Dit betekent dat we een regel nodig hebben voor de interactie van waaiers met verschillende uitgebreide labels.



(240)

Aangezien de waaiers op elkaar en niet op een grens inwerken, legt de regel, net als de regel voor de interactie van een applicator en een waaier (217) met een uitgebreid label het verwisselen van de volgorde van de knopen vast. De uitgebreide labels van de waaiers blijven hetzelfde.

Als de waaiers die in de gebogen tak terechtkomen hetzelfde uitgebreide label hebben (232) als de waaiers die dit paar omvatten, dan verstoren de nieuwe waaiers het evenwicht in de tak niet. Omdat de m-waaiers (239) het evenwicht in de tak wel zouden verstoren, zwakken we de opvatting van evenwicht van waaiers (186) af.

Definitie: **waaier-gebalanceerdheid** (241)

Stelt s een abstractor of een $(0,1)$-grens, x steeds een willekeurige variabele, $-$ een grens, ϵ de lege expressie, en $\triangle$ een waaier voor die niet naar s wijst, en kun je voor iedere s met de regels

$$\begin{aligned} e_0 &= \epsilon \mid @e_0 \mid \triangle e_0 \mid \lambda e_0 \mid \lambda x\, e_0 \mid -e_0 \mid e_1 e_0 \\ e_1 &= \triangle u_{(n,0)} e_0 \triangle v_{(0,n)} \end{aligned}$$

de opeenvolging van knopen in de tak beschrijven, zodanig dat $\triangle u$ een paar vormt met $\triangle v$ en beide naar s wijzen, dan houden de waaiers in die tak elkaar in evenwicht. □

De definitie kwantificeert over verwijzingen naar abstractors en grenzen. Dit betekent dat we voor iedere abstractor kijken of de waaiers in de tak die naar die abstractor wijzen zich samen met de rest van de knopen in de tak laten weergeven zoals de clausules dat voorschrijven.

We nemen aan dat we bij het toepassen van de clausules in de definitie kunnen bepalen of twee waaiers een paar vormen of niet. Wat de definitie betreft is het vormen van een paar dus impliciet.

Wanneer we door de m-waaier lopen mogen we niet de naam van de stapel halen die we er bij de n-waaier op leggen. Omdat we niet alleen de naam van een poort, maar daarnaast ook het uitgebreide label van de waaier op de stapel leggen, kunnen we, wanneer we door de m-waaier lopen, naar een naam zoeken die bij deze waaier past.

Naast de naam die we zoeken hoeft niet dezelfde uitbreiding te liggen als die van het label van de waaier waar we doorheen lopen. Want tussen die waaier en de abstractor of $(0,1)$-grens waar deze naar wijst bevinden zich mogelijk nog andere abstractors of grenzen. De volgende definitie houdt hier rekening mee.

Definitie: **het effect van een waaier op de stapel** (242)

Met $x \in V$, $s_1, s_2 \in S$, $s_2 \neq \epsilon$, $p \in \{1,2\}$ en $n_1, n_2 \in \mathbb{N}$, leggen

$$\begin{aligned} n_1 \neq n_2 \;\wedge\; s_1 = (s_2, \langle p, n_2 \rangle) &\;\rightarrow\; \triangle_-(\langle n_1, s_1 \rangle) = \langle p, \triangle_-(\langle n_1, s_2 \rangle)\rangle \\ n_1 \neq n_2 \;\wedge\; s_1 = (s_2, \langle p, n_2, \bowtie \rangle) &\;\rightarrow\; \triangle_-(\langle n_1, s_1 \rangle) = \langle p, \triangle_-(\langle n_1, s_2 \rangle)\rangle \\ \\ n_1 \neq n_2 \;\wedge\; s_1 = (s_2, x) &\;\rightarrow\; \triangle_-(\langle n_1, s_1 \rangle) = \triangle_-(\langle n_1 - 1, s_2 \rangle) \\ n_1 \neq n_2 \;\wedge\; s_1 = (s_2, \langle x, \bowtie \rangle) &\;\rightarrow\; \triangle_-(\langle n_1, s_1 \rangle) = \triangle_-(\langle n_1 - 1, s_2 \rangle) \end{aligned}$$

$$
\begin{array}{lcll}
n_1 \neq n_2 & \wedge & s_1 = (s_2, \lightning) & \rightarrow \quad \triangle_{-}(\langle n_1, s_1 \rangle) = \triangle_{-}(\langle n_1 - 1, s_2 \rangle) \\
n_1 \neq n_2 & \wedge & s_1 = (s_2, \langle \lightning, \bowtie \rangle) & \rightarrow \quad \triangle_{-}(\langle n_1, s_1 \rangle) = \triangle_{-}(\langle n_1 - 1, s_2 \rangle)
\end{array}
$$

$$
\begin{array}{lcll}
n_1 = n_2 & \wedge & s_1 = (s_2, \langle p, n_2 \rangle) & \rightarrow \quad \triangle_{-}(\langle n_1, s_1 \rangle) = \langle p, \triangle_{-}(\langle n_1, (s_2, \langle p, n_2, \bowtie \rangle) \rangle) \rangle \\
n_1 = n_2 & \wedge & s_1 = (s_2, \langle p, n_2, \bowtie \rangle) & \rightarrow \quad \triangle_{-}(\langle n_1, s_1 \rangle) = \langle p, \triangle_{-}(\langle n_1, (s_2, \langle p, n_2, \bowtie \rangle) \rangle) \rangle
\end{array}
$$

een afbeelding $\triangle_{-} : \mathbb{N} \times S \to \{1, 2\} \times S$ vast, een afbeelding die de eerste zichtbare naam op de stapel, de naam van een poort die bij een waaier hoort waarvan n het label uitbreidt, laat zien en definitief onzichtbaar maakt. □

Het eerste paar clausules geeft aan dat er met de naam van een poort die bij een waaier hoort waarvan niet n_1 maar een andere waarde het label uitbreidt niets gebeurt. Als je die naam kan zien, blijft ze zichtbaar, en een naam die je niet kan zien blijft onzichtbaar.

Het tweede en het derde paar clausules leggen vast dat als we een $\lightning$ of de naam van een variabele tegenkomen, we verder gaan met zoeken. De zichtbaarheid van die naam of $\lightning$ verandert niet. Omdat we aan de andere kant van een abstractor verdergaan, moeten we naar waaiers kijken waarvan de uitbreiding van het label één kleiner is.

De zevende clausule geeft weer dat de naam die bij een waaier hoort, verdwijnt, dat wil zeggen: de afbeelding maakt de naam met een $\bowtie$ definitief onzichtbaar. Omdat de naam niet boven op de stapel, maar ook tussen andere namen in kan liggen, blijft de naam op of in de stapel liggen.

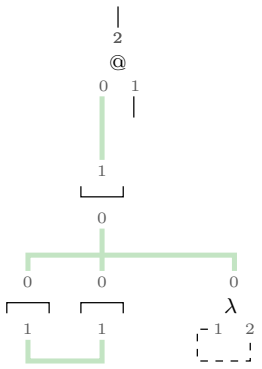
De achtste clausule geeft aan dat we, wanneer we een niet zichtbare naam tegenkomen, verder moeten zoeken.

Omdat de afbeelding $\triangle_{-}$ alleen met stapels die bij gebalanceerde takken horen doet wat we ervan verwachten, kunnen we $\triangle_{-}$ niet als een totale afbeelding opvatten.

Net als $\rightarrowtail$ en $\triangle_{\bowtie}$ gebruiken we de afbeelding $\triangle_{-}$ in onderdeel 6 in de definitie van de afbeelding van grafen op expressies. Omdat we het op de stapel leggen van de naam van een poort als een beperkte handeling zien, leggen we het effect daarvan niet met een aparte afbeelding vast.

De interactie tussen grenzen die geen van beide een uitgebreid label hebben

Als de 0-tak van een applicator met een grens begint, er na die grens twee grenzen volgen die een paar vormen, en er daarna een abstractor volgt, dan lezen we een expressie met een abstractie en een applicatie, terwijl de grenzen de interactie tussen de applicator en de abstractor in de weg zitten.



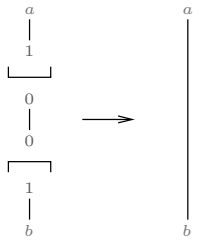
(243)

Zou de grens die de aftakking aangeeft de grens aan het begin van de gebogen tak opheffen, dan brengt dat de applicator en de abstractor dichter bij elkaar.

Wanneer we eerst een $(1, 0)$-grens en daarna een $(0, 1)$-grens oversteken, halen we de poorten van de stapel die bij waaiers horen die buiten de gebogen tak geen paar vormen met een andere waaier. Verder vervangt een $\lightning$ de naam van de variabele die bij de gebogen tak hoort. Na de interactie blijven allen namen liggen.

Omdat de waaiers in de tak na de tweede grens elkaar in evenwicht houden, maakt het echter geen verschil of de namen van de poorten die we vóór de interactie van de stapel halen na de interactie blijven liggen. Bovendien halen we ze samen met de $\lightning$ bij de derde grens weg, zodat we na die grens de graaf weer met dezelfde stapel lezen.

Het wegvallen van de grenzen verandert de expressie die we lezen niet, en daarom lijkt de volgende regel geschikt.

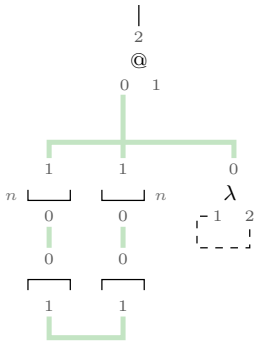


(244)

Hoewel bij het wegvallen van de grenzen de context soms wel en soms niet op dezelfde manier verandert, steken we, afgezien van het paar grenzen dat op elkaar inwerkt, vóór en na de interactie in de rest van de tak dezelfde grenzen in dezelfde richting over. De interactie verstoort het evenwicht van grenzen zonder (199) en grenzen met een uitgebreid label (208) dus niet.

De interactie tussen een grens met en een grens zonder uitgebreid label

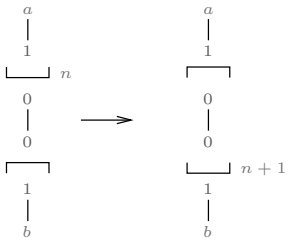
Als er in de 0-tak van een applicator een paar grenzen met een uitgebreid label zit, met tussen die grenzen een paar grenzen zonder uitgebreid label, dan lezen we een expressie met een abstractie in een applicatie, terwijl die paren de abstractor in de 0-tak van de applicator voorafgaan.



(245)

Wanneer we een grens met een uitgebreid label zouden verwisselen met een grens zonder uitgebreid label, dan brengt dat de ene grens dichter bij de applicator en de andere dichter bij de abstractor.

Als we bij het zichtbaar of onzichtbaar maken van namen naar een naam of ↯ zoeken, en daarna een ↯ op de stapel leggen, zouden we ook eerst die ↯ kunnen toevoegen. We moeten dan bij het zoeken wel één naam of ↯ verder kijken. De context verandert dan op dezelfde manier. Omdat we bovendien de knopen die we veranderen niet in de expressie terugzien, lezen we hetzelfde. Regels van de volgende vorm lijken daarom geschikt.



(246)

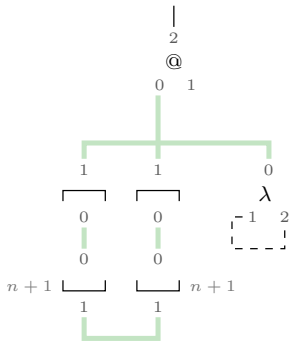
Ook in het geval waarin we de tweede grens zonder uitgebreid label verwisselen met de tweede grens met een uitgebreid label, volgt dat we één bij de uitbreiding van het label van de waaier op moeten tellen.

De interactie verstoort het evenwicht van de grenzen niet. Want na de interactie kan je wat de definitie (208) betreft de eerste en de tweede grens bij elkaar laten horen. De derde

grens hoort dan bij de abstractor of grens zonder uitgebreid label waar de eerste grens bij hoorde. Wat de vierde grens betreft hoeft er niets te veranderen.

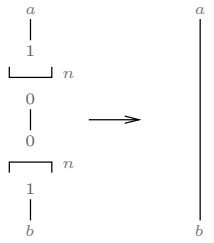
De interactie tussen grenzen, beide met hetzelfde uitgebreide label

Met het verwisselen van de grenzen (245) naderen de grenzen met het uitgebreide label elkaar.



(247)

Omdat met het verdwijnen van de grenzen de applicator en de abstractor elkaar naderen, de ene grens het effect van de andere teniet doet, en we geen van beide terugzien in de expressie, lijken regels die de grenzen opheffen geschikt.

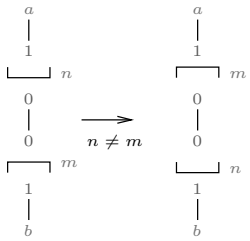


(248)

Als de grenzen in takken in de graaf vóór de interactie van de grenzen elkaar in evenwicht houden, dan doen ze dat na de interactie ook: wanneer je na de interactie die definitie op de tak toepast, kun je de stap waarin je vóór de interactie de clausule kiest die bij de grenzen hoort, gewoon achterwege laten.

De interactie van grenzen met verschillende uitgebreide labels

Omdat waaiers grenzen voor zich uit duwen (205) brengt de interactie van waaiers met verschillende uitgebreide labels (240) ook de interactie van grenzen met verschillende labels met zich mee.



(249)

Zodra een grens met een uitgebreid label tussen grenzen in een gebogen tak terechtkomt, verstoort dit het evenwicht (208) van grenzen. Naast het evenwicht van waaiers moeten we ook het evenwicht van grenzen verzwakken.

Definitie: **grens-gebalanceerdheid** (250)

Stelt s een abstractor of een $(0,1)$-grens, x steeds een willekeurige variabele, $\triangle$ een waaier, ϵ de lege expressie, en $-$ een grens die niet naar s hoort, en kun je voor iedere s met de regels

$$\begin{aligned} e_0 &= \epsilon \mid @e_0 \mid \triangle e_0 \mid \lambda e_0 \mid \lambda x\, e_0 \mid -e_0 \mid e_1 e_0 \\ e_1 &= -u_{(n,0)} e_0 - v_{(0,n)} \end{aligned}$$

de opeenvolging van knopen in de tak beschrijven, zodanig dat beide grenzen $-u$ en $-v$ bij s horen, dan houden de grenzen in die tak elkaar in evenwicht. □

De definitie heeft betrekking op alle verwijzingen van knopen in de tak naar abstractors of $(0,1)$-grenzen in de graaf. Als we de grenzen die bij ieder van die abstractors horen op de manier weer kunnen geven die clausules voorschrijven, dan houden de grenzen in de tak elkaar in evenwicht.

De tweede clausule beschrijft grenzen, mogelijk met verschillende uitgebreide labels, die bij één en dezelfde abstractor horen. We nemen aan dat we op de één of andere manier kunnen achterhalen of twee grenzen bij dezelfde abstractor horen of niet.

Omdat het evenwicht van waaiers en het evenwicht van grenzen deel uitmaken van de definitie van de invariant, krijgen we met de afgezwakte vormen van het evenwicht ook een nieuwe invariant.

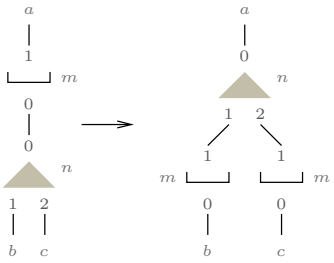
Definitie: **o-invariant** (251)

De voorafgaande invariant (218) waarin de definities gebalanceerdheid (186, 208) plaats maken voor de nieuwe opvatting van het evenwicht van waaiers (241) en het evenwicht (250) van grenzen. □

Wanneer we alle o-regels op een rijtje hebben, laten we zien dat de interactie zich aan deze invariant houdt.

De interactie tussen waaiers en grenzen, beide met een uitgebreid label

Door op een applicator in te werken kunnen waaiers en grenzen die een uitgebreid label hebben zich tussen de waaiers en grenzen in een gebogen tak begeven. Omdat een grens die een uitgebreid label heeft op die manier een waaier tegen kan komen die ook een uitgebreid label heeft, hebben we de volgende regel nodig.



(252)

Met de situatie waarin waaiers en grenzen met een uitgebreid label op elkaar inwerken, hebben we alle mogelijke combinaties van waaiers en grenzen bekeken. We laten alleen nog zien dat we geen regels nodig hebben voor de interactie van twee applicators of twee abstractors.

Situaties met twee abstractors

Omdat iedere abstractor met de 0-poort naar de wortel van de graaf wijst, verwachten we niet dat we een regel nodig hebben voor de interactie van twee abstractors.

Stelling: **abstractors en interactie** (253)

Een abstractor zit nooit met haar 0-poort aan de 0-poort van een andere abstractor vast.

Bewijs: een abstractor wijst met de 0-poort naar de wortel van de graaf. Met inspectie van de o-regels volgt dat interactie dit niet verandert. Een abstractor kan alleen inwerken op

knopen in de tak die naar die abstractor leidt. Zit er een abstractor in die tak, dan wijst deze ook met haar 0-poort naar de wortel, en daardoor kan de ene abstractor niet op de andere inwerken. □

Vanwege het bewijs volgt dat we geen interactieregel nodig hebben die betrekking heeft op twee abstractors.

Situaties met twee applicators

Niet alleen een abstractor, maar ook een applicator wijst met haar poorten altijd in dezelfde richting. We verwachten daarom niet dat we een regel nodig hebben die beschrijft hoe de ene op de andere applicator inwerkt.

Stelling: **applicators en interactie** (254)

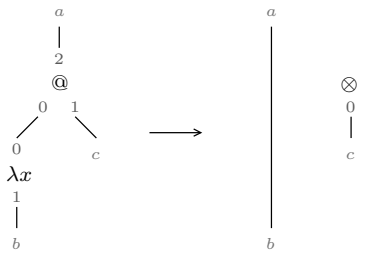
Een applicator zit in geen enkele o-graaf met haar 0-poort aan de 0-poort van een andere applicator vast.

Bewijs: het bewijs van de stelling (253) maar dan met de constatering dat de applicator in de tak met haar 0-poort naar de wortel wijst. □

Met het bewijs van de stelling volgt dat we ook geen regel nodig hebben voor de interactie van twee applicators.

De wisser

De regel (198) voor de interactie van een applicator en een abstractor heeft betrekking op een abstractor die een gebogen tak heeft. Als een abstractor geen gebogen tak heeft, bindt ze geen enkele variabele. Werkt een applicator op een dergelijke abstractor in, dan komt deze interactie niet met substitutie overeen.

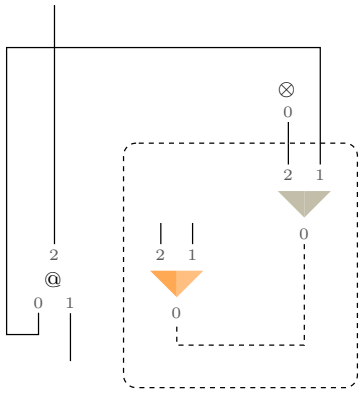


(255)

We lezen de 1-tak van de applicator niet meer, en daarom laten we deze, net als in het voorbeeld dat we op p.56 geven, met een wisser beginnen.

Anders dan een abstractor met een gebogen tak heeft een abstractor die deze niet heeft een uitgebreid label. Het label bevat namelijk de naam van de variabele die de abstractor zou binden. Dit betekent dat we de weergave (255) alleen als een schema op mogen vatten: voor iedere uitbreiding hebben we een dergelijke regel.

Omdat we na de interactie in de kant waar de 1-tak van de applicator mee begon niet meer verder lezen, zou de wisser knopen in die tak weg kunnen halen.



(256)

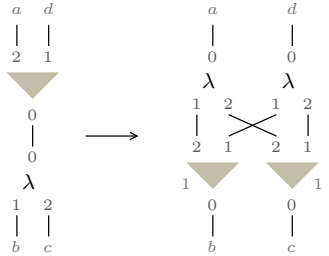
Maar we moeten opletten. Want het fragment dat de wisser weg zou poetsen, kunnen we ook via een andere tak lezen. Een waaier voegt in dat geval de takken samen. De wisser moet vóór de waaier stoppen. Want de waaier mag vanwege het evenwicht in de takken niet verdwijnen. Behalve de regel voor de interactie van een applicator en een abstractor met twee poorten, geven we daarom geen regels waar een wisser in voorkomt.

Alle regels bij elkaar

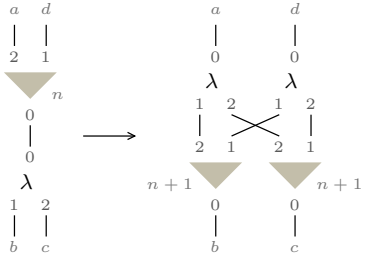
Op zoek naar de regels die de interactie van een applicator en een abstractor mogelijk maken, hebben we voorbeelden gevonden van samenstellingen waaruit de noodzaak van een regel blijkt. Van de overige situaties hebben we laten zien (219, 235, 253, 254) dat ze zich niet voordoen.

We geven een overzicht van de regels die we nodig hebben. Merk op dat de index van een waaier één of groter, en die van een grens twee of groter is.

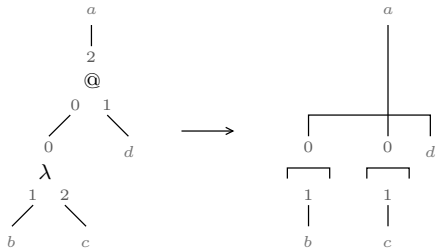
Naast de interactie van een abstractor die drie poorten heeft, geven we ook de regels voor de interactie voor abstractors die maar twee poorten hebben. Dergelijke abstractors mogen we voor wat de interactie betreft als transparant opvatten.



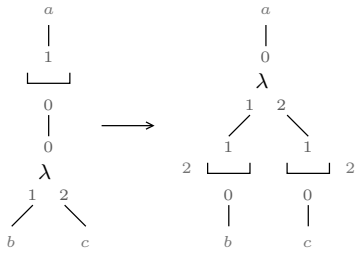
$\triangle\lambda$ (191)



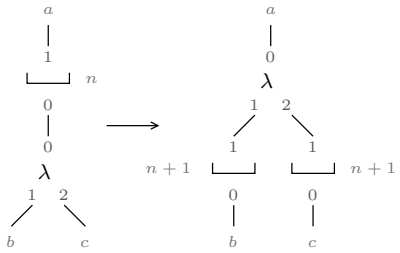
$\triangle n\lambda$ (192)



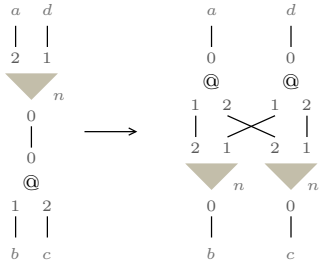
$@\lambda$ (198)



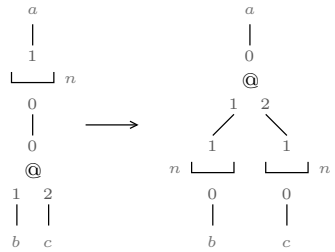
$-\lambda$ (206)



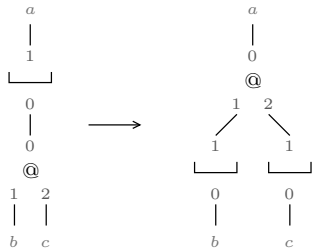
$-n\lambda$ (207)



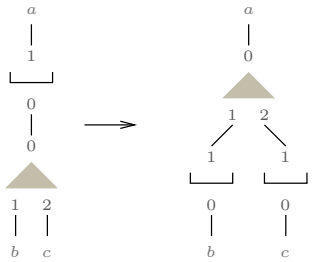
$\triangle n@$ (217)



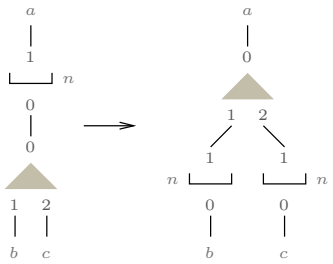
$-n@$ (224)



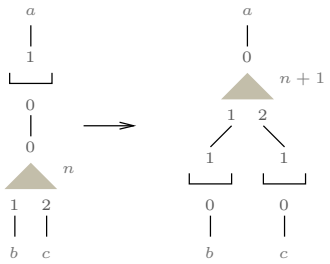
$-@$ (227)



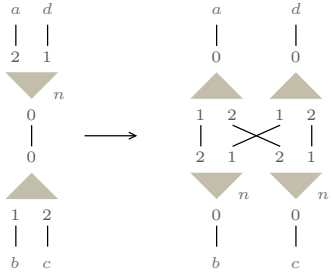
$-\triangle$ (229)



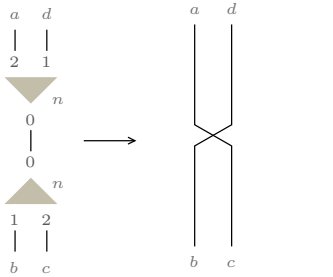
$-n\triangle$ (231)



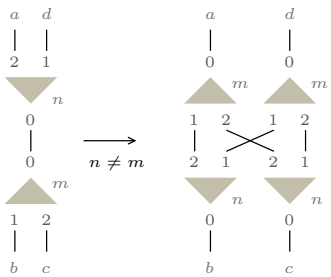
$-\triangle n$ (233)



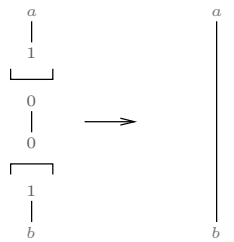
$\triangle n\triangle$ (237)



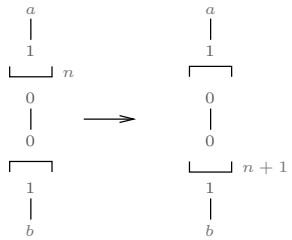
$2\triangle n$ (238)



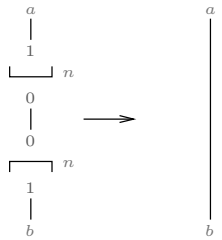
$\triangle n\triangle m$ (240)



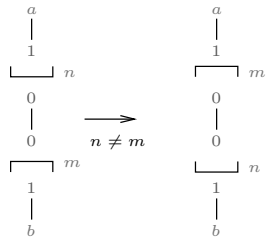
$2-$ (244)



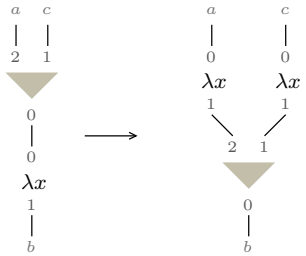
$-n-$ (246)



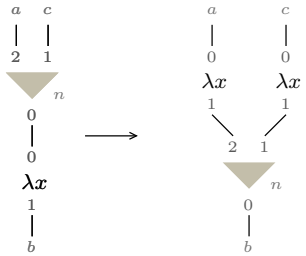
$2-n$ (248)



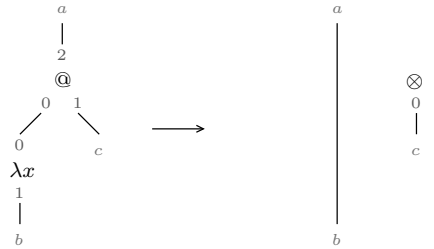
$-n-m$ (249)



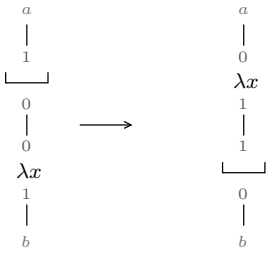
$\triangle\lambda$ (257)



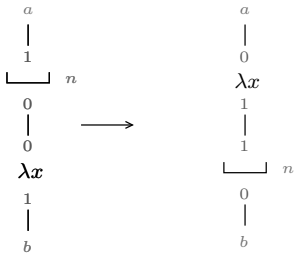
$\triangle n\lambda$ (258)



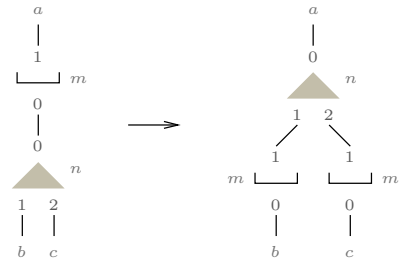
$@\lambda$ (255)



$-\lambda$ (259)



$-\lambda$ (260)



$\triangle n - m$ (252)

De definitie van interactie

Wanneer we een voorbeeld vinden waarin een knoop met een uitgebreid label een rol speelt, dan vinden we eigenlijk niet één maar meerdere voorbeelden. Omdat er op de eerste grens in het paar dat bij de interactie van een grens en een abstractor ontstaat weer een abstractor kan volgen, kun je voor een grens met een willekeurige uitbreiding een voorbeeld vinden.

Stelling: **uitgebreide labels van grenzen** (261)

Als je een regel nodig hebt voor de interactie van een bepaalde knoop en een grens met een uitgebreid label, dan heb je voor iedere grens met een uitbreiding $n > 1$ een regel nodig voor de interactie met die knoop.

Bewijs: kijk naar een situatie met een grens met een uitgebreid label. Vanwege de interactie van een abstractor en een grens kun je de situatie vervangen door een nieuwe waarin de uitbreiding van het label van de grens één groter is. □

Niet alleen voor de interactie met een grens met een uitgebreid label hebben we oneindig veel regels nodig.

Stelling: **uitgebreide labels van waaiers** (262)

Als je een regel nodig hebt voor de interactie van een bepaalde knoop en een waaier met een uitgebreid label, dan heb je voor iedere waaier met een uitbreiding $n > 1$ een regel nodig voor de interactie met die knoop.

Bewijs: het bewijs van de stelling (261) voor grenzen, maar dan met een vervanging van de waaier. □

Op basis van de schema's voor de regels leggen we met de structuren (49, 55) die we in onderdeel 2 hebben besproken de interactie vast die hetzelfde effect zou moeten hebben als de reductie van λ-expressies: eerst leggen we de verzameling regels vast, daarna geven we aan hoe een interactienet eruitziet, en tenslotte volgt de definitie van de interactie.

Definitie: **o-regels** (263)

Met iedere n, $n > 0$ in het geval van een waaier, en $n > 1$ bij een grens, geeft ieder schema (p.151, 152, en 153) aanleiding tot een interactieregel. □

Naast regels bestaat een interactienet uit een graaf. Het uitgebreide label van knopen geven we de vorm van een paar: de eerste component bevat het oorspronkelijke label, de tweede bevat de uitbreiding.

Definitie: **o-interactienet** (264)

Met

- $\mathcal{L}$ de verzameling labels

$$\{\odot, @, \lambda, \langle\lambda, x\rangle, \triangle, \langle\triangle, n_1\rangle, -, \langle -, n_2\rangle \mid n_1 > 0, n_2 > 1\}$$

- $\alpha \subset \mathcal{L} \times \mathbb{N}$ de relatie

$$\begin{aligned}&\{\langle\odot, 1\rangle, \langle @, 3\rangle, x\\ &\langle\lambda, 3\rangle, \langle\langle\lambda, x\rangle, 2\rangle,\\ &\langle\triangle, 3\rangle, \langle\langle\triangle, n_1\rangle, 3\rangle,\\ &\langle -, 2\rangle, \langle\langle -, n_2\rangle, 2\rangle \mid x \in V, n_1 > 0, n_2 > 1\}\end{aligned}$$

- g een $\mathcal{L}\alpha$-graaf, en

- R de verzameling o-regels,

is het geordende paar $\langle g, R\rangle$ een o-interactienet. □

Stelling: **robuustheid** (265)

De definitie van o-interactienet is robuust.

Bewijs: R is deterministisch en voor iedere $r \in R$ is $\Pi_3(r)$ een bijectie. □

Met het interactienet waarmee je begint, en een afspraak over de manier waarop het ene net in het andere verandert, ligt interactie vast. In het geval van de interactie die hetzelfde effect heeft als de reductie van een λ-expressie, een o-interactienet, beginnen we altijd met een e-normaalvorm.

Definitie: **o-interactie** (265)

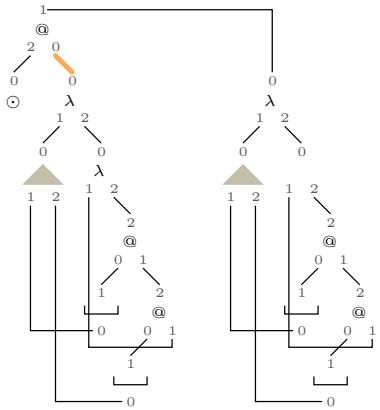
Met een e-normaalvorm g_1 en R de verzameling o-regels, is o-interactie in het net $\langle g_1, R\rangle$ de opeenvolging van stappen die met

$$\langle\langle g_1, R\rangle, \langle g_2, R\rangle\rangle$$

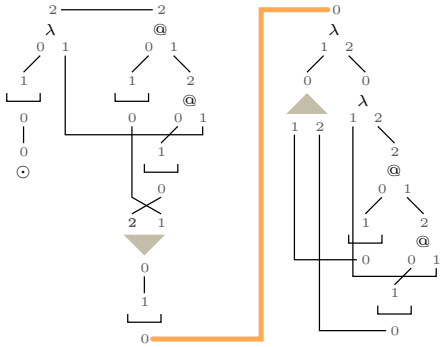
begint. □

Een voorbeeld

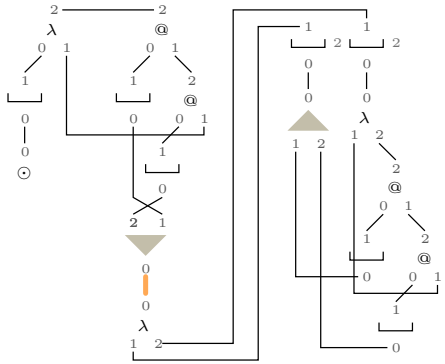
Bij wijze van voorbeeld kijken we naar een interactienet waarin we de Church-weergave van 2 toegepast op zichzelf lezen: een applicator waarvan de takken bestaan uit twee instanties van de normaalvorm in het voorbeeld (p.100) van e-interactie.



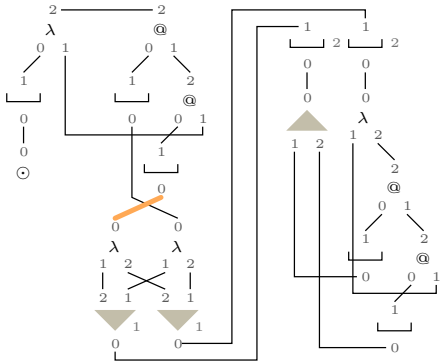
@λ, 198



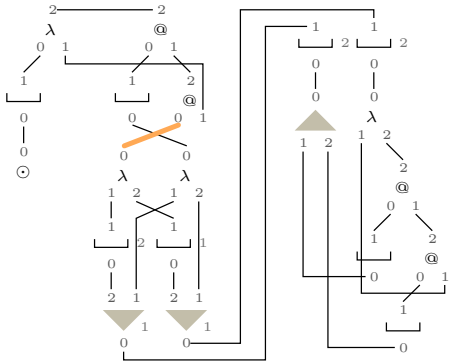
−λ, 206



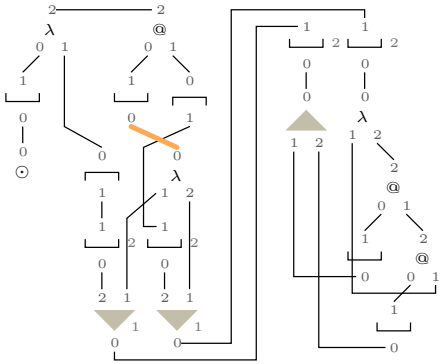
△λ, 191



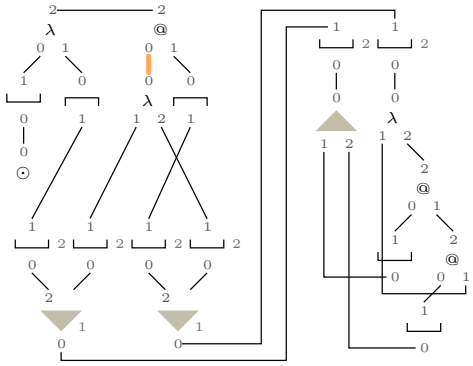
−nλ, 207



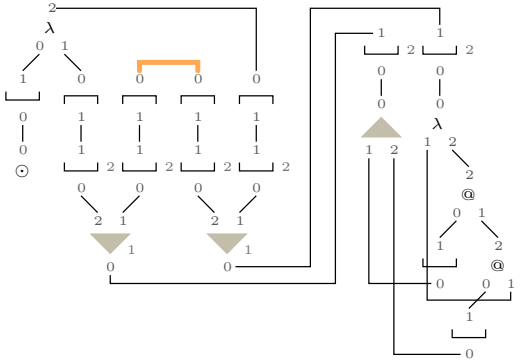
@λ, 198



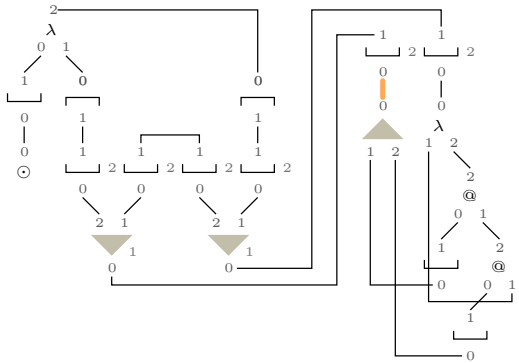
−λ, 206



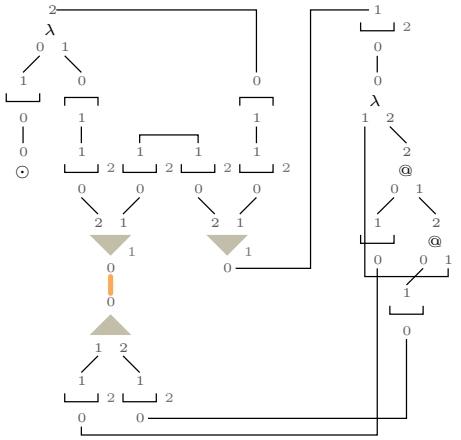
@λ, 198



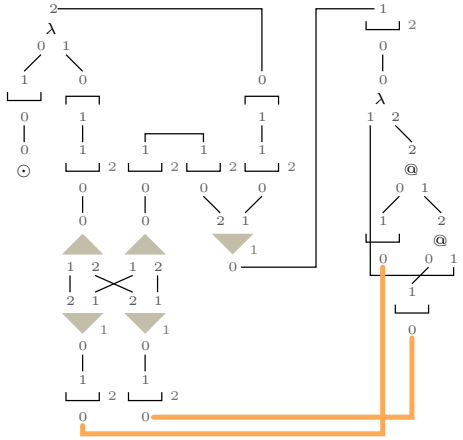
2−, 244



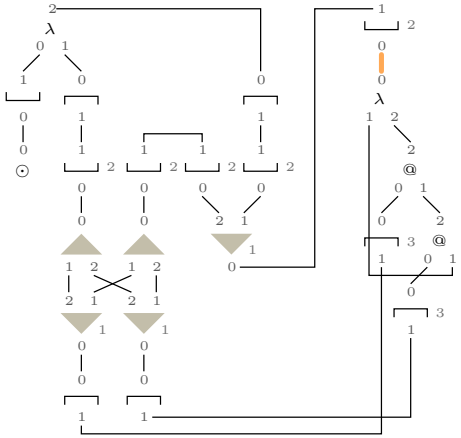
$-n\triangle,\ 231$



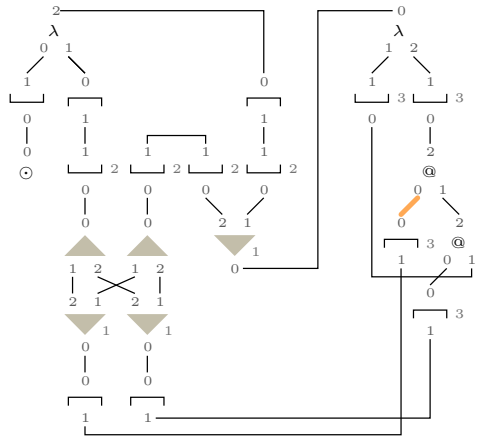
$\triangle n\triangle,\ 237$



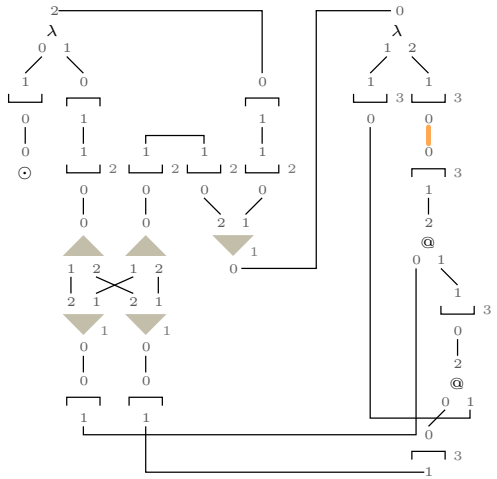
$-n-,\ 246$



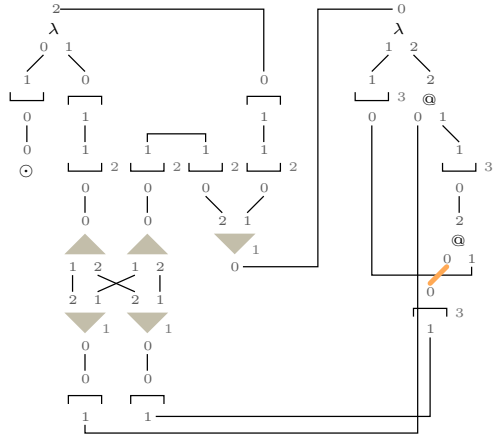
$-n\lambda,\ 207$



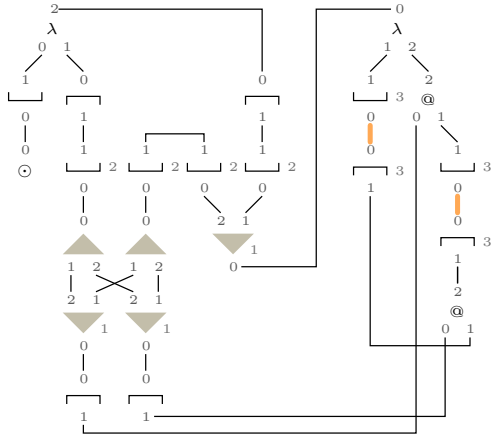
$-n@,\ 224$



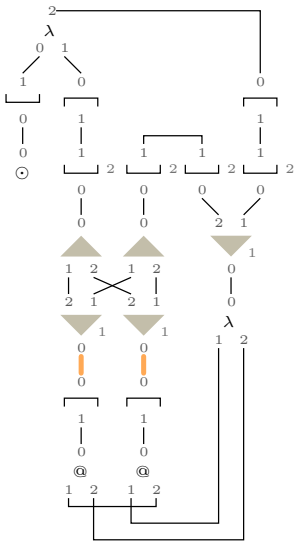
$2-n,\ 248$



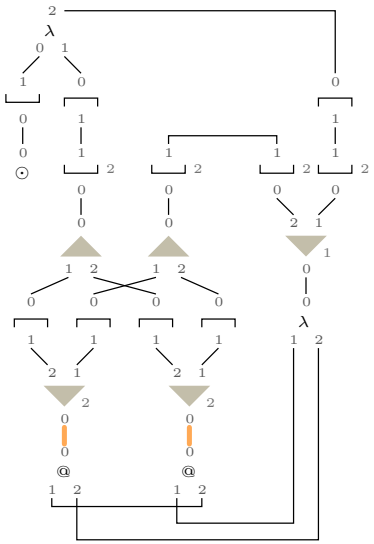
$-n@,\ 224$



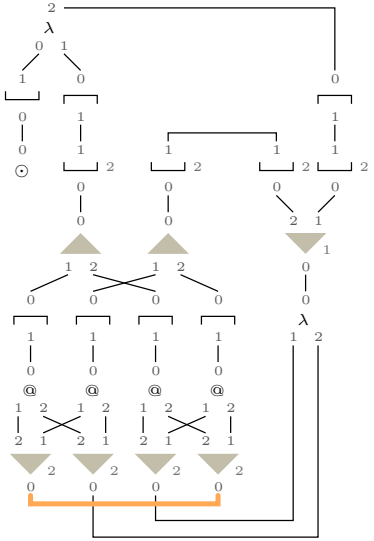
$2-n,\ 248$



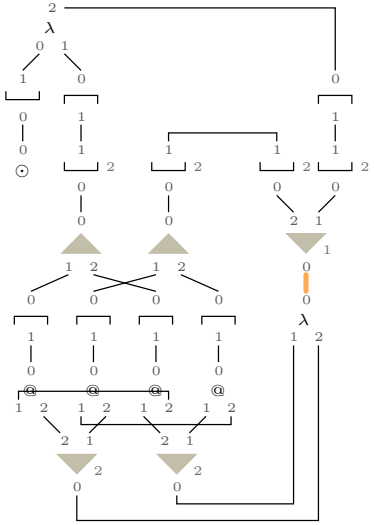
$-\triangle n$, 233



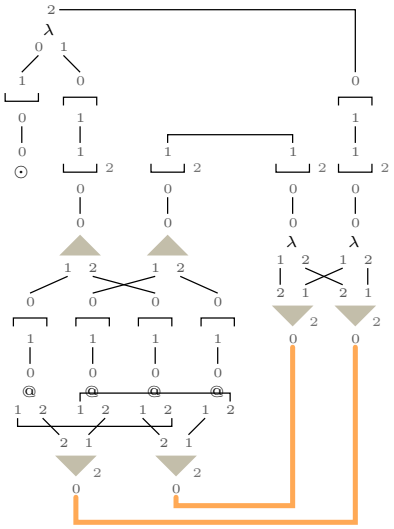
$\triangle n@$, 217



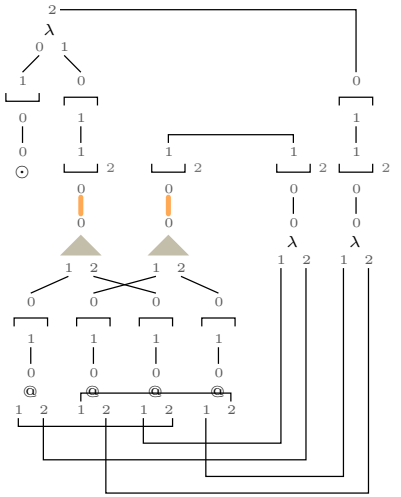
$2\triangle n$, 238



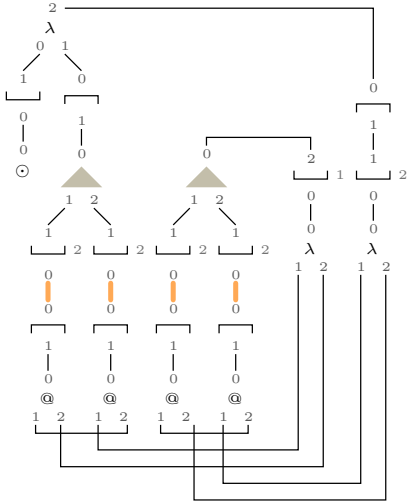
$\triangle n\lambda$, 192



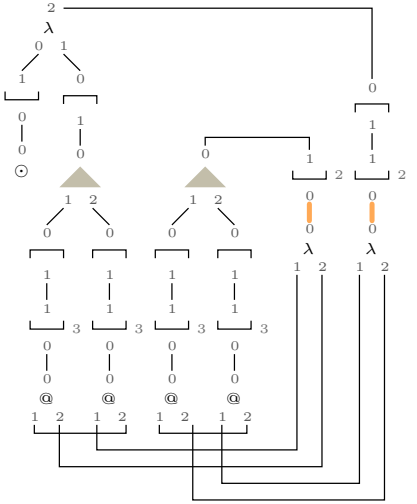
$2\triangle n$, 238



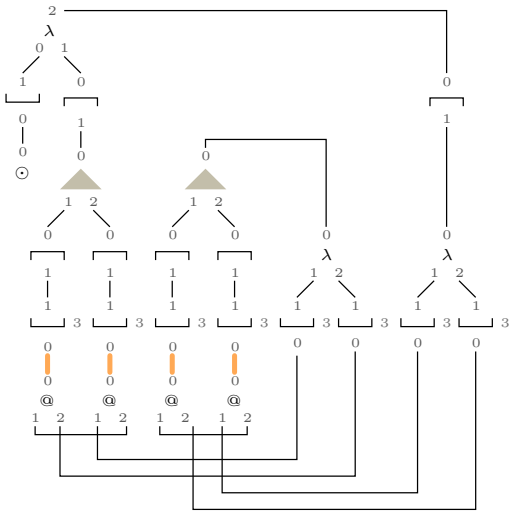
$-n\triangle$, 231



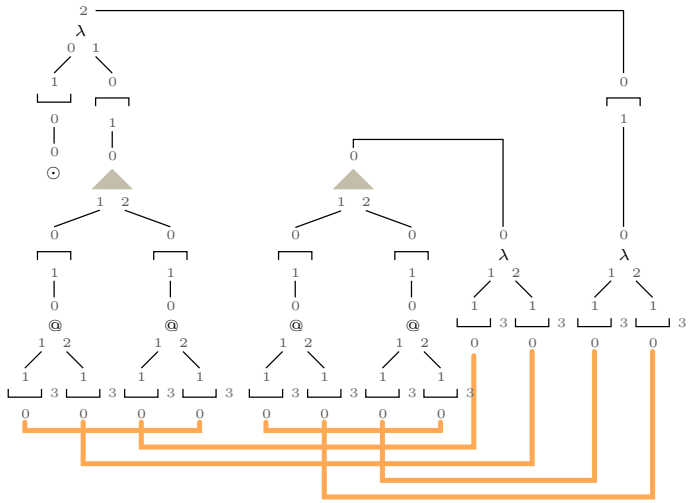
$-n-$, 246



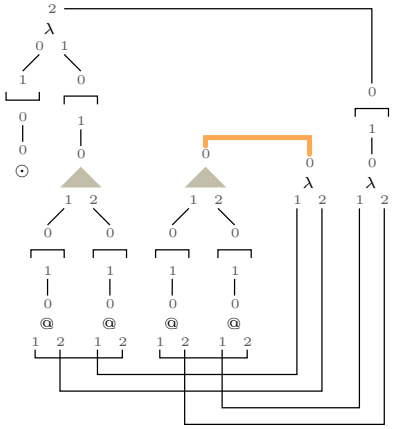
$-n\lambda$, 207



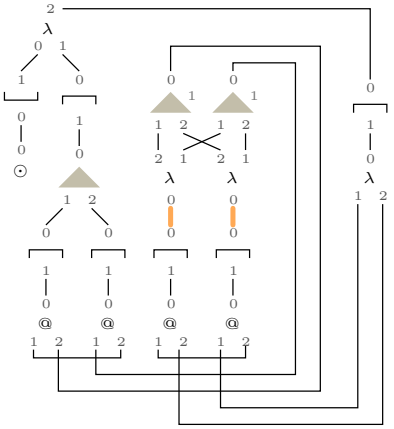
$-n@,\ 224$



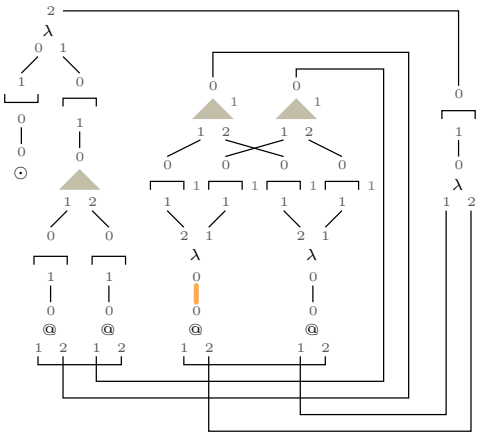
$2-n,\ 248$



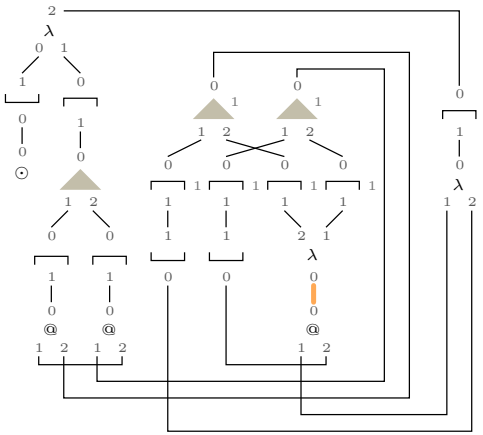
$\triangle\lambda,\ 191$



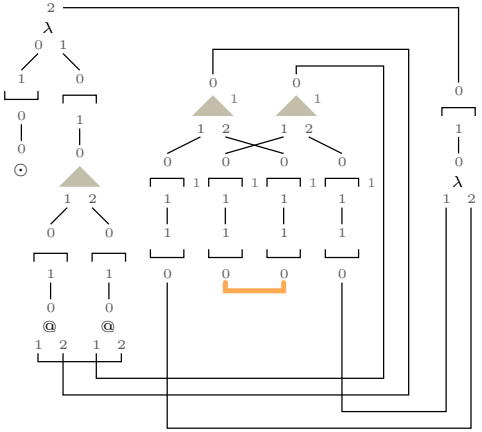
$-\lambda,\ 206$



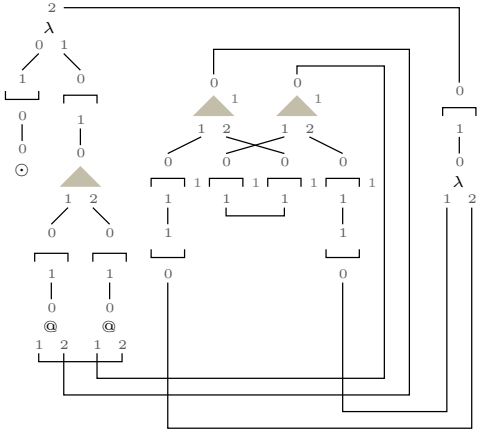
@λ, 198



@λ, 198



2−, 244



In de graaf waarmee we beginnen lezen we $\underline{2}$ toegepast $\underline{2}$ en daarom zouden we in de normaalvorm

$$\begin{aligned}
& (\lambda v.\lambda w.v(vw))\ (\lambda x.\lambda y.x(xy)) \\
\rightarrow_\beta\ & \lambda w.(\lambda x.\lambda y.x(xy))\ (\ (\lambda x.\lambda y.(x(xy)))\ w) \\
\rightarrow_\beta\ & \lambda w.(\lambda x.\lambda y.x(xy))\ (\lambda y.(w(wy))) \\
\rightarrow_\beta\ & \lambda w.\lambda y.(\lambda y.(w(wy)))\ ((\lambda y.(w(wy)))\ y) \\
\rightarrow_\beta\ & \lambda w.\lambda y.w(w\ ((\lambda y.(w(wy)))\ y)\ ) \\
\rightarrow_\beta\ & \lambda w.\lambda y.w(w\ (w(wy))\ )
\end{aligned}$$

moeten lezen. Dit is de Church-weergave van vier. De volgorde van de stappen in de reductie ligt niet vast: in de vierde stap mogen we ook eerst de y substitueren. Als de interactie in de graaf de reductie in de rekenkunde volgt, dan moeten we deze expressie in de normaalvorm (p.163) lezen. Die lijkt op het eerste gezicht echter niet op de syntaxboom die $\underline{4}$ weergeeft.

Maar in het fragment tussen de tweede abstractor en de eerste applicator, het fragment met de grenzen en de waaiers, lezen we niets, en hoeven we alleen maar de naam van de poort waardoor we de waaier verlaten te onthouden. Dit maakt het een beetje gemakkelijker. Op weg naar de eerste applicator nemen we de 1-poort van de 1-waaier.

Na de applicator lezen we in de 0-tak na de grenzen en de waaier de variabele die we bij de eerste abstractor op de stapel leggen. Nemen we w voor die variabele en kiezen we y in de tweede abstractor, dan lezen we

$$\lambda w.\lambda y.w...$$

De 1-tak van de eerste applicator leidt naar de tweede applicator, en ook in de 0-tak van die applicator lezen we weer dezelfde variabele

$$\lambda w.\lambda y.w(w...)$$

De 1-tak van de tweede applicator leidt weer naar de waaiers. We nemen de 1-poort, en verlaten het fragment met de waaiers via de 2-poort van de 1-waaier. We komen opnieuw in de applicator uit, en dat betekent dat we de variabele w nog twee keer lezen.

$$\lambda w.\lambda y.w(w(w(w...)))$$

Wanneer we weer bij de waaiers komen, nemen we de 2-poort zodat we uiteindelijk aan het einde van de gebogen tak van de tweede abstractor y lezen. We lezen in de graaf dus toch dezelfde expressie als de reductie in de λ-rekenkunde oplevert.

Interactie houdt zich aan de invariant

De interactie in een o-graaf laat niet alleen een applicator op een abstractor inwerken, ze maakt ook ruimte voor dergelijke interactie. Daarbij mag de expressie die we in de graaf lezen niet veranderen. De interactie moet het evenwicht van de waaiers en dat van de abstractors en grenzen bewaren, ze mag de richting waarin de poorten van een applicator of een abstractor wijzen niet veranderen, en ze moet er voor zorgen dat het ene paar waaiers of grenzen het andere omvat.

Na de laatste uitbreiding (218) van de invariant, en na het vastleggen van de interactie, laten we zien dat de interactie zich aan de invariant houdt.

Stelling: **o-invariantie** (266)

De grafen in o-interactienetten maken de invariant (251) waar.

Bewijs: neem een interactieregel, en veronderstel dat het linkerdeel deel uitmaakt van een graaf die de beweringen in de invariant waar maakt. Laat zien dat een stap die de regel volgt tot een graaf leidt die de beweringen nog steeds waar maakt. Doe dit voor iedere o-regel in het interactienet.

Neem bijvoorbeeld de regel (217) voor de interactie van een applicator en een waaier met een uitgebreid label. Stel je voor dat een gebogen tak via de 0-poort van een applicator loopt, en de tweede waaier in een paar zich bij de 0-poort meldt. Vanwege het evenwicht van waaiers (241) hoeft de eerste waaier in dat paar niet op de voorafgaande abstractor te volgen.

De waaier die na de interactie aan de applicator voorafgaat, houdt daar de eerste waaier in het paar in evenwicht. Met de interactie verschijnt er in de 1-tak van de applicator ook een waaier. Deze waaier heft het effect van de waaier die inmiddels aan de applicator voorafgaat weer op.

Als ook de 1-tak van de applicator deel uitmaakt van de gebogen tak, dan volgt er vanwege het evenwicht (241) in die tak nog een tweede waaier. De waaier die in de 1-tak verder gaat vormt een paar met die waaier. Met de interactie ontstaat er in dit geval een extra paar waaiers in de tak. Het nieuwe paar waaiers verstoort het evenwicht niet. □

Volledigheid

Omdat met o-invariantie niet volgt dat er tussen een applicator en een abstractor geen enkele waaier of grens zit, kunnen we alleen op basis van de invariant niet laten zien dat de interactie in een graaf ervoor zorgt dat geen enkele waaier of grens de interactie van een applicator en een abstractor in de weg zit.

Zouden we een graaf afbeelden op de som van de afstanden van applicators en abstractors die op elkaar in kunnen werken, dan daalt de waarde van de afbeelding niet bij iedere stap in de interactie. Denk bijvoorbeeld maar aan de interactie (229) van een waaier en een grens. Deze interactie brengt de applicator en de abstractor niet dichter bij elkaar.

Maar interactie maakt de afstand van een waaier of grens tot de applicator of abstractor die ze met haar 0-poort aanwijst wel kleiner. Wanneer we voor iedere waaier of grens tussen een applicator en een abstractor deze afstand meetellen, maakt interactie tussen waaiers en grenzen onderling de som van deze afstanden wel kleiner.

Definitie: **variante functie** (267)

De variante functie voor o-interactie voegt voor een bepaalde applicator de som van de bijdragen van de waaiers en grenzen aan een o-graaf toe.

- Wijst een grens of een waaier met haar 0-poort naar de 0-poort van een applicator, dan draagt ze haar afstand tot die applicator bij aan de waarde van de functie.
- Waaiers en de grenzen die met hun 0-poort naar de eerste abstractor in de 0-tak van de applicator wijzen, dragen hun afstand tot die abstractor bij.

□

Als de som van de afstanden gelijk is aan nul, zit geen enkele waaier of grens de interactie van de applicator en de abstractor in de weg.

Stelling: **volledigheid** (268)

De variante functie voor o-interactie heeft een nulpunt.

Bewijs: voor iedere knoop die met haar 0-poort aan de 0-poort van een andere knoop vastzit, hebben we een interactieregel. Zit een regel met een bepaalde combinatie van knopen niet in de verzameling, dan doet de situatie zich ook niet voor (219, 235, 253, 254). Met inspectie van de o-regels volgt dat bij iedere interactie de waarde van de functie daalt. Omdat we voor iedere mogelijke situatie een regel geven, volgt dat de waarde tot nul daalt. □

Ruimte alleen is niet genoeg

Met de regels die we hebben vastgelegd maakt de interactie altijd ruimte voor een optimale stap, de interactie van een applicator en een abstractor. Maar dit wil nog niet zeggen dat de interactie de graaf op een zodanige manier verandert dat wat we in de normaalvorm lezen hetzelfde is als de expressie die een reductie in de λ-rekenkunde op zou leveren.

Wanneer we willen nagaan hoe interactie een graaf verandert, moeten we eerst vastleggen hoe we een graaf lezen. We weten dat we een graaf in een context lezen, en daarom kunnen we het effect van de verschillende knopen op de context vastleggen. Daarnaast hebben we een raamwerk nodig waarin we grafen met elkaar kunnen vergelijken. Bij het vergelijken hebben we de invariant (166, 218) weer nodig.

6 Grafen lezen

Hoewel lokale binding grafen van syntaxbomen doet verschillen, moet interactie in grafen uiteindelijk wel hetzelfde effect hebben als de stappen in de λ- rekenkunde. Als we bij een verandering in een graaf geen reductie van een λ- expressiekunnen vinden, dan werkt de interactie niet; ze wijkt af van wat er in de rekenkunde gebeurt.

Afgezien van de interactie tussen een applicator en een abstractor hebben we geprobeerd interactie de expressie die we lezen niet te laten veranderen. Omdat we daarbij alleen naar een fragment kijken, en niet verder dan één symbool, kunnen we eigenlijk nog niets zeggen over de expressie die we in de graaf lezen.

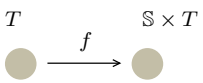
Het lastige in het vergelijken van grafen heeft dus niet alleen met gebogen takken of de stapel te maken. Want ook bij het vergelijken van twee syntaxbomen zouden we niet genoeg hebben aan het vergelijken van fragmenten. We gaan na hoe we bomen of grafen met elkaar moeten vergelijken.

Stap voor stap, symbool na symbool, toestand na toestand

Omdat we niet in één oogopslag zien wat we in een graaf lezen, moeten we er stap voor stap doorheen lopen. Daarbij onthouden we in ieder geval de kant waar we staan. Zie je het lezen van een graaf als een proces, dan vormt de graaf zelf, de kant, en wat je verder eventueel nog moet onthouden, de toestand waarin dit proces zich bevindt.

Nog afgezien van verschillen voor wat kanten en knopen betreft, zetten we in de grafen die we moeten vergelijken niet altijd dezelfde stappen. Want bij de eerste stap in het fragment waar interactie betrekking op heeft, kan de toestand na de interactie op een andere manier veranderen als vóór de interactie.

Zouden we niet naar de tussenliggende waaiers en grenzen kijken, maar alleen naar de knopen die een symbool als label hebben, dan moeten we in grafen die we vergelijken dezelfde opeenvolging van symbolen zien. We leggen het lezen daarom vast als een bewerking die een toestand op een volgende toestand en op een knoop met een label afbeeldt.



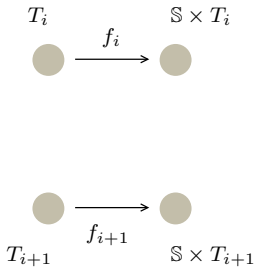
(269)

Als we zeggen dat we een expressie in een graaf lezen, dan betekent dit eigenlijk dat we een opeenvolging van symbolen afzonderen. Omdat we een opvolging van symbolen los zien van de toestanden, vatten we een stap waarin we een symbool lezen daarom op als een bewerking in een co-algebra, een algebra met twee opeenvolgingen.

Co-algebra's vergelijken

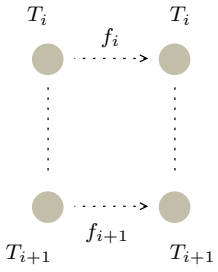
Als we in twee grafen dezelfde expressie lezen, dan brengen de co-algebra's die bij het lezen horen dezelfde opeenvolging van symbolen met zich mee. De opeenvolging van toestanden hoeft niet hetzelfde te zijn. Denk bijvoorbeeld maar aan equivalente grafen: we lezen hetzelfde terwijl de knopen en kanten verschillen.

Omdat opeenvolgingen van toestanden mogen verschillen, hoeft ook de verzameling van toestanden die bij het lezen van een graaf hoort niet dezelfde te zijn. Alleen in dit opzicht verschilt de bewerking in de co-algebra's die bij het lezen horen. We lezen grafen natuurlijk wel altijd op dezelfde manier.



(270)

Bij equivalente grafen horen co-algebra's die dezelfde opeenvolging van symbolen met zich meebrengen. Maar de toestanden hoeven niet hetzelfde te zijn, en daarom hebben we bij het lezen van equivalente grafen mogelijk ook met verschillende co-algebra's te maken. Wanneer je vanaf het begin steeds een stap in de ene en daarna een stap in de andere zet, komen de toestanden waarin je terechtkomt echter op een bepaalde manier met elkaar overeen.



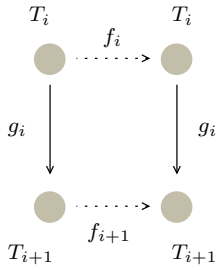
(271)

Kijken we niet naar equivalente grafen, maar naar grafen vóór en na een interactie, een interactie die de opeenvolging van symbolen die we lezen niet verandert, dan komen de toestanden ook met elkaar overeen. Weliswaar op een andere manier.

Weet je op welke manier de toestanden van elkaar mogen verschillen, dan kun je nagaan of een interactie de opeenvolging van symbolen verandert. Als je het gebied waar interactie

betrekking op heeft in beide grafen in een overeenkomstige toestand begint te lezen, en de toestand komt na het gebied nog steeds overeen, dan verandert de opeenvolging niet.

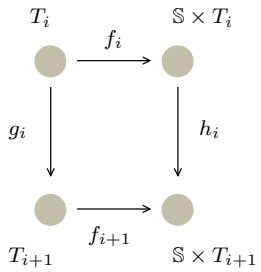
We zoeken dus een relatie op de verzameling toestanden van graaf co-algebra's. Wanneer je twee grafen met elkaar vergelijkt, kijken we naar twee toestanden, en daarom hebben we genoeg aan een afbeelding.



(272)

Als het lezen van de graaf na de interactie een opeenvolging van toestanden met zich meebrengt die met de afbeelding g_i samenvalt, dan verandert die opeenvolging op de manier die we voor ogen hebben. De co-algebra's die bij het lezen van de graaf horen brengen in dat geval dezelfde opeenvolging van symbolen met zich mee.

Omdat de interactie de opeenvolging van symbolen niet mag veranderen, mogen we niet vergeten naast de toestanden ook de symbolen te vergelijken. Hiervoor gebruiken we de afbeelding die symbolen op zichzelf afbeeldt, en de ene aan de andere toestand toevoegt zoals g_i dat doet.



(273)

We weten nog niet hoe g_i en h_i eruitzien. Maar als we het beeld van een symbool of een toestand in de eerste co-algebra willen vergelijken met een symbool of een toestand in de tweede co-algebra, dan moet voor de afbeeldingen en de bewerkingen in de co-algebra's in ieder geval gelden dat

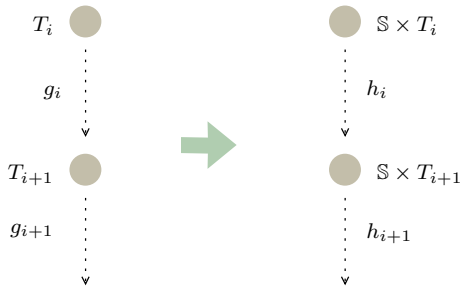
$$f_{i+1} \circ g_i = h_i \circ f_i$$

We zeggen ook wel dat het schema (273) moet commuteren.

Een speciale toevoeging

Wanneer we willen nagaan of het schema (273) commuteert, moeten we niet alleen de bewerking in de co-algebra's vastleggen, maar ook de toestanden in de ene co-algebra met die in de andere verbinden. We kijken eerst naar de manier waarop we het domein en het bereik van de verschillende bewerkingen in de co-algebra's in kaart brengen.

Omdat interactie grafen verandert, krijgen we bij iedere volgende graaf die we lezen met een andere verzameling toestanden te maken. Met die verzamelingen als objecten, en de afbeeldingen van de ene naar de andere verzameling als pijlen, brengt interactie een categorie, een verzameling objecten en pijlen, met zich mee.



(274)

Vanwege het feit dat het lezen van een graaf een co-algebra met zich meebrengt, en de bewerking daarin naast elementen van T_i ook op het symbolen afbeeldt, hebben we niet met één maar naar twee categorieën te maken: de bewerking in een co-algebra voegt aan een T_i in de ene categorie een $I_{\mathbb{S}} \times T_i$ in de andere toe.

Hoewel we niet weten hoe de pijlen in de categorieën eruitzien, verschilt een pijl in de tweede categorie niet veel van een pijl in de eerste. Terwijl $I_{\mathbb{S}} \times T_i$ wel het domein van h_i is, kunnen we niet zeggen dat

$$h_i = I_{\mathbb{S}} \times g_i$$

want

$$\{\langle\langle s, s\rangle, \langle t, g_i(t)\rangle\rangle \mid s \in \mathbb{S} \wedge t \in T_i\}$$

beeldt niet een symbool en een toestand in de ene co-algebra op een symbool en een toestand in de andere co-algebra af. Nemen we echter

$$\{\langle\langle s, t\rangle, \langle s, g_i(t)\rangle\rangle \mid s \in \mathbb{S} \wedge t \in T_i\}$$

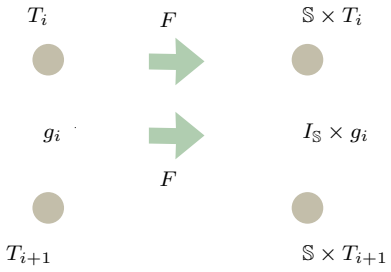
als alternatieve opvatting van het product van de twee afbeeldingen, dan kunnen we wel zeggen dat

$$h_i = I_{\mathbb{S}} \times g_i$$

Met X de objecten en pijlen in de ene categorie en $\times$ een combinatie van de gebruikelijke en alternatieve opvatting van het product van verzamelingen voegt

$$F : X \to \mathbb{S} \times X$$

de objecten en pijlen in de ene categorie toe aan die van de andere.



(275)

Met de bewerking in de co-algebra die bij het lezen van een graaf hoort, voegt F aan een verzameling T_i een verzameling

$$\mathbb{S} \times T_i$$

toe. Daarnaast voegt ze met de alternatieve opvatting van het product aan een afbeelding g_i de afbeelding

$$I_{\mathbb{S}} \times g_i$$

toe.

Omdat de F zich op andere afbeeldingen baseert, noemen we haar vanwege Carnap's perspectief op het toevoegen van eigenschappen [14, p.13-14] ook wel een functor. Wat hij hij een functor noemde, voegt niet zoals een predicaat rechtstreeks, maar via een functie een eigenschap aan een object toe.

Vanwege het feit dat de interactie een graaf op verschillende manieren in één bepaalde graaf kan veranderen, en er bij iedere graaf een verzameling toestanden hoort, kunnen we in de categorie ook via verschillende opeenvolgingen van pijlen van het ene naar het andere object gaan.

Als twee opeenvolgingen van pijlen in de eerste categorie (275) in één domein eindigen, dan moet de F een pijl in de eerste categorie toevoegen aan een pijl in de tweede categorie, zodanig dat de opeenvolgingen van pijlen de tweede categorie ook in hetzelfde bereik eindigen. Met andere woorden: de toevoeging F moet compositioneel zijn.

De theorie dwingt het compositionele af door in de definitie van een functor [15, p.13] een clausule op te nemen die aangeeft dat het eerst samenstellen van pijlen en daarna afbeelden op hetzelfde neerkomt als het samenstellen van de beelden van pijlen.

Met de interactie liggen de verzamelingen van toestanden en de opeenvolging van pijlen vast die een verzameling met een volgende verbindt; interactie legt een categorie vast. Met

de functor ligt ook het domein en het bereik vast van de bewerkingen in de co-algebra's die bij het lezen van een graaf horen.

We moeten nog nagaan hoe een pijl die de ene met de andere verzameling verbindt er precies eruitziet. De functor beperkt in ieder geval de afbeeldingen waar we naar zoeken, want $F(g_i)$ neemt de plaats van h_i in. We zoeken naar één afbeelding, een homomorfisme van co-algebra's.

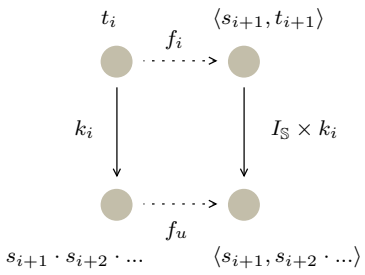
Uiteindelijke co-algebra's

Wanneer je de bewerkingen in twee co-algebra's toepast, en daardoor stap voor stap in de toestanden die bij die co-algebra's horen terechtkomt, kun je op een gemakkelijke manier een homomorfisme van de ene naar de andere co-algebra vastleggen: beeldt na iedere stap de toestand in de ene co-algebra af op die in de andere.

We willen echter nagaan of een interactie in een willekeurige graaf de expressie die we lezen niet verandert. Omdat we tenslotte niet door alle mogelijke co-algebra's die bij een interactie horen kunnen lopen, moeten we de afbeelding van de ene naar de andere graaf co-algebra op een andere manier proberen te vinden.

Omdat de functor F niet alleen verzamelingen met toestanden die bij het lezen van grafen horen kan afbeelden, hoef je je helemaal niet tot een categorie te beperken waarvan de objecten uit verzamelingen van dergelijke toestanden bestaan; we proberen twee graaf co-algebra's via een andere, co-algebra [5, p.24] met elkaar te verbinden.

Wanneer we de eerste co-algebra (273) via een co-algebra die bij de functor F hoort op de tweede co-algebra afbeelden, hoeven we voor wat betreft het vinden van de beelden van de symbolen in ieder geval niet door de eerste co-algebra te lopen; vanwege de functor beeldt de identiteit de symbolen af.



(276)

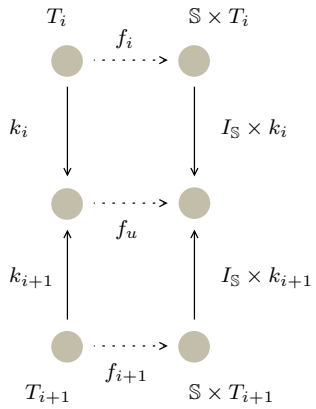
Ook wat het afbeelden van toestanden betreft hoef je geen stappen te zetten. Stel jezelf in plaats daarvan voor dat k_i de symbolen die f_i in t_i nog zou genereren aan t_i toevoegt. Het beeld van t_i bestaat dan uit de denkbeeldige opeenvolging die met s_{i+1} begint. Neem je de kop van deze lijst als het volgende symbool, en de staart als de volgende toestanden,

dan ligt f_u vast, terwijl het schema (276) commuteert.

Omdat een toestand in de nieuwe co-algebra alleen van de volgorde van de symbolen afhangt, kan die co-algebra niet met verschillende opeenvolgingen van toestanden toch dezelfde opeenvolging van symbolen met zich meebrengen. Dit betekent dat er maar één homomorfisme van een co-algebra naar de nieuwe kan bestaan.

De nieuwe co-algebra geeft weer waar het in de co-algebra's die je erop afbeeldt afgezien van de bewerking om gaat, namelijk: de opeenvolging van de symbolen. We noemen de co-algebra daarom ook wel een uiteindelijke co-algebra. Het uiteindelijke houdt niet in dat je de co-algebra niet meer op een andere af zou kunnen beelden.

Zolang we dezelfde opeenvolging symbolen lezen, verbindt één en dezelfde uiteindelijke co-algebra de co-algebra's die bij het lezen horen. Verandert de opeenvolging, dan krijgen we met een andere uiteindelijke co-algebra te maken.



(277)

Hoewel er in de vorm van de uiteindelijke co-algebra wel een verbinding tussen de ene en de andere graaf co-algebra bestaat, brengt de uiteindelijke co-algebra niet de afbeelding die we zoeken. Want vanuit die co-algebra zie je de toestanden in de graaf co-algebra's niet, en kun je $(k_{i+1})^{-1}$ en daarmee ook $(k_{i+1})^{-1} \circ k_i$ niet vastleggen.

Bisimulatie

De uiteindelijke co-algebra verbindt twee graaf co-algebra's (277) met elkaar, zonder dat we daarbij te weten komen bij welke toestanden in de tweede graaf co-algebra de toestanden in de eerste graaf co-algebra horen. We proberen daarom één co-algebra te vinden waaraan je kunt zien welke toestanden in de twee graaf co-algebra's bij elkaar horen.

Omdat we een toestand in de ene graaf co-algebra in verband willen brengen met een toestand in de andere graaf co-algebra, laten we de toestanden van de nieuwe co-algebra

uit paren bestaan. Nemen we de toestanden die bij elkaar horen als componenten, dan raken we de verbinding met de uiteindelijke co-algebra echter kwijt. We nemen daarom niet de toestanden zelf, maar hun beelden (276) in de uiteindelijke co-algebra: als voor $t_1 \in T_i$ en $t_2 \in T_{i+1}$ geldt dat

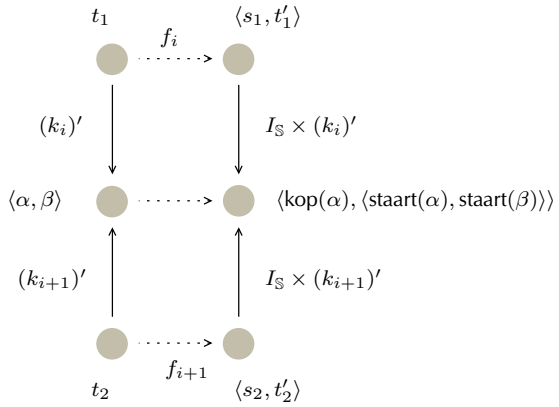
$$k_i(t_1) = k_{i+1}(t_2)$$

dan is

$$\langle k_i(t_1), k_{i+1}(t_2) \rangle$$

de nieuwe toestand.

Aangezien een toestand in de nieuwe co-algebra [5, p.24] uit een paar van opeenvolgingen van symbolen bestaat, kost het niet veel moeite de bewerking vast te leggen: beeldt, als de opeenvolging uit meer dan één symbool bestaat, het paar op een nieuwe toestand af, een paar dat uit de staarten van beide opeenvolgingen bestaat. Omdat we met homomorfismen afbeelden, kun je voor het volgende symbool de kop van één van beide nemen.



(278)

De bewerking in de nieuwe co-algebra genereert niet alleen een symbool en een volgende toestand. Omdat de toestanden uit paren bestaan legt ze ook een relatie R vast. Als je met een paar begint, dan genereert de bewerking het volgende paar, en het volgende, net zo lang totdat de opeenvolging uit één symbool bestaat. Voor opeenvolgingen α en β geldt

$$R(\alpha, \beta) \rightarrow R(\text{staart}(\alpha), \text{staart}(\beta))$$

Wanneer je veronderstelt dat beide graaf co-algebra's dezelfde opeenvolging van symbolen met zich meebrengen, dan volgt vanwege de manier waarop we de bewerking in de nieuwe co-algebra hebben vastgelegd, dat deze bij dezelfde uiteindelijke co-algebra hoort als die welke bij de graaf co-algebra's hoort.

Omdat de co-algebra's allemaal bij dezelfde uiteindelijke co-algebra horen, bestaan er ook homomorfismen $(k_i)'$ en $(k_{i+1})'$ van de graaf co-algebra's naar de nieuwe. Met R bestaat er

daarom ook een relatie B op de toestanden van de graaf co-algebra's. Een relatie waarvoor geldt dat

$$B(t_1, t_2) \to B(f_i(t_1), f_{i+1}(t_2))$$

De andere kant op gaat ook: neem aan dat er een relatie B op de toestanden van twee graaf co-algebra's bestaat waarvoor de voorafgaande implicatie geldt. Wanneer je de toestanden in de graaf co-algebra's op de componenten van de toestanden van de nieuwe co-algebra afbeeldt zoals je ze op de bijbehorende uiteindelijke co-algebra af zou beelden, en je

$$\langle \alpha, \beta \rangle$$

als toestand krijgt, dan bestaat de volgende toestand in de nieuwe co-algebra vanwege de relatie B uit twee staarten

$$\langle \text{staart}(\alpha), \text{staart}(\beta) \rangle$$

Als je veronderstelt dat de bewerkingen in de beide graaf co-algebra's hetzelfde symbool genereren,

$$B(t_1, t_2) \to \pi_1(f_i(t_1)) = \pi_1(f_{i+1}(t_2))$$

oftewel

$$\text{kop}(\alpha) = \text{kop}(\beta)$$

en je dit symbool als volgende symbool in de nieuwe co-algebra neemt, dan hebben de graaf co-algebra's en de nieuwe co-algebra vanwege het commuteren van de schema's (278) dezelfde uiteindelijke co-algebra waardoor

$$\text{staart}(\alpha) = \text{staart}(\beta)$$

zodat

$$\alpha = \beta$$

We hebben het volgende resultaat.

Stelling: **bisimulatie** (279)

Een co-algebra met de bewerking

$$f_i : T_i \to F(T_i)$$

brengt dan en slechts dan dezelfde opeenvolging van symbolen met zich mee als een co-algebra met de bewerking

$$f_{i+1} : T_{i+1} \to F(T_{i+1})$$

als er een relatie

$$B \subseteq T_i \times T_{i+1}$$

bestaat waarvoor

$$B(t_1, t_2) \rightarrow \pi_1(f_i(t_1)) = \pi_1(f_{i+1}(t_2)) \wedge B(\pi_2(f_i(t_1)), \pi_2(f_{i+1}(t_2)))$$

Alleen als twee co-algebra's dezelfde opeenvolging van symbolen met zich meebrengen kan een relatie bestaan die aan de implicatie voldoet. De bewerkingen in de co-algebra's doen elkaar in dat geval na, en het is daarom dat we die relatie een bisimulatie noemen.

Geen lijsten maar bomen

Omdat we bij het lezen in een graaf bij een applicator niet in één maar in twee toestanden verder gaan, krijgen we eigenlijk helemaal geen lijst met symbolen, maar een boom. De bewerking die bij het lezen van een graaf hoort, beeldt een toestand op nul, één of twee nieuwe toestanden af.

We lezen syntaxbomen, en omdat de takken in dergelijke bomen een bepaalde volgorde hebben, beeldt de bewerking in de co-algebra's een toestand een geordende lijst van nul, één of twee toestanden af.

Als een syntaxboom eindig is, kom je in iedere tak in een toestand terecht waarin je niet verder kunt; er volgt geen nieuw symbool en ook geen nieuwe toestand. We moeten de bewerking in de co-algebra die bij het lezen van een eindige graaf hoort dus als een partiële functie opvatten.

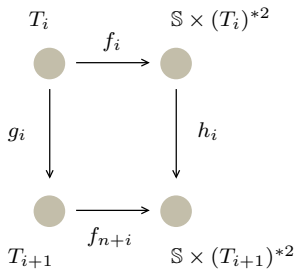
Definitie: **De bewerking in graaf co-algebra's** (280)

Met de verzameling T_i van toestanden, en $\mathbb{S}$ de verzameling symbolen is

$$f_i : T_i \rightarrow \mathbb{S} \times (T_i)^{*2}$$

het voorschrift van de bewerking in een graaf co-algebra. □

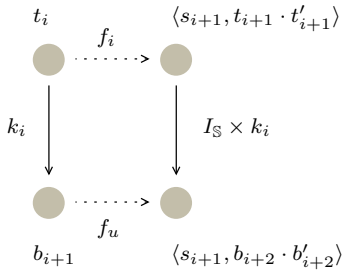
Wanneer we de ene op de andere graaf co-algebra afbeelden, dan voegen we na de stap vanwege de bewerking de ene geordende lijst aan de andere toe.



(281)

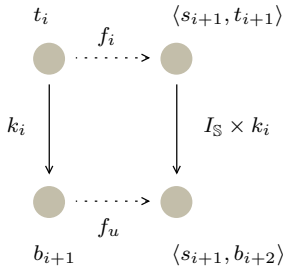
Bij de lijst co-algebra's commuteert het schema niet als de beelden niet kloppen. Omdat de bewerking in een graaf co-algebra partieel is, kan er bij dergelijke co-algebra's een tweede reden bestaan waarom een schema niet commuteert: de bewerking in de ene co-algebra beeldt de toestand wel af, de bewerking in de andere niet.

Ook graaf co-algebra's brengen we met elkaar in verband door een co-algebra vast te leggen waarin iedere toestand bestaat uit dat wat we in die toestand nog moeten lezen, namelijk een boom. Als een boom vóór een stap uit een wortel met twee takken bestaat, dan vinden de we de linker tak en de rechter tak na de stap in een geordende lijst van bomen terug:



(282)

Als er maar één tak verder gaat, komt alleen die tak in het beeld van de bewerking in de tweede co-algebra terecht.



(283)

Als we in een toestand van een graaf co-algebra niet verder gaan, dan doen we dat in de nieuwe co-algebra ook niet.

De bewerking in nieuwe co-algebra genereert een boom met symbolen. Omdat die boom verandert zodra er iets in de toestanden verandert, kan er van een co-algebra die dezelfde boom met zich meebrengt naar de nieuwe co-algebra maar één homomorfisme bestaan; de nieuwe co-algebra is dus een uiteindelijke.

Zou er vanuit iedere knoop altijd maar één tak verder gaan, dan legt de bewerking dezelfde relatie vast als de bewerking in lijst co-algebra's:

$$R(\alpha, \beta) \to R(\mathsf{tak}(\alpha), \mathsf{tak}(\beta))$$

Als er meer dan één tak vertrekt moeten we ook meerdere takken met elkaar in verband brengen. Vertrekken er takken in α en β dan

$$R(\alpha,\beta) \rightarrow \#\text{takken}(\alpha) = \#\text{takken}(\beta) \bigwedge_{j=0}^{\#\text{takken}(\alpha)-1} R(\text{takken}(\alpha)(j), \text{takken}(\beta)(j))$$

Alleen als hetzelfde aantal takken uit een knoop vertrekt, kunnen we iedere tak met een andere in verband brengen. Verder moeten we op de volgorde van de takken letten: we brengen de ene linker tak met de andere linker tak, en de ene rechter tak met de andere rechter tak in verband.

Vergelijkbaar met de afleiding van bisimulatie voor lijsten (279) volgt op basis van de veronderstelling dat als twee co-algebra's dezelfde boom met zich meebrengen, de relatie R en de manier waarop je de graaf co-algebra's de nieuwe co-algebra afbeeldt, dat op die toestanden van de graaf co-algebra's een relatie B bestaat waarvoor, als t_1 en t_2 een beeld hebben met

$$\gamma_1 = \pi_2(f_i(t_1))$$

en

$$\gamma_2 = \pi_2(f_{i+1}(t_2))$$

geldt dat

$$B(t_1,t_2) \rightarrow \#\gamma_1 = \#\gamma_2 \bigwedge_{j=0}^{\#\gamma_1-1} R(\gamma_1(j), \gamma_2(j))$$

Voegen we de gelijkheid

$$\pi_1(\gamma_i) = \pi_1(\gamma_{i+1})$$

van de symbolen na de stappen in de co-algebra's aan de implicatie toe, dan volgt, ook weer op een vergelijkbare manier, dat de beide graaf co-algebra's dezelfde boom met zich meebrengen.

Stelling: **bisimulatie** (284)

Een co-algebra met de bewerking

$$f_i : T_i \rightarrow F(T_i)^{*2}$$

brengt dan en slechts dan dezelfde syntaxbomen met zich mee als een co-algebra met de bewerking

$$f_{i+1} : T_{i+1} \rightarrow F(T_{i+1})^{*2}$$

als er een relatie

$$B \subseteq T_i \times T_{i+1}$$

op de toestanden van de graaf co-algebra's bestaat waarvoor, als t_1 en t_2 een beeld hebben, met lijsten

$$\gamma_i = f_i(t_1)$$

en

$$\gamma_{i+1} = f_{i+1}(t_2)$$

geldt dat

$$B(t_1, t_2) \rightarrow \pi_1(\gamma_i) = \pi_1(\gamma_{i+1}) \wedge \#(\gamma_i) = \#(\gamma_{i+1}) \wedge \bigwedge_{j=0}^{\#(\gamma_1)-1} B(\pi_2(\gamma_i)(j), \pi_2(\gamma_{i+1})(j))$$

Bij het lezen en vergelijken van grafen hebben we twee co-algebra's, de eerste hoort bij het lezen van de graaf vóór de interactie, de tweede bij de graaf na de interactie. Als je een bisimulatie op de verzameling van toestanden van de co-algebra's vast kunt leggen, dan verandert de interactie de boom die de co-algebra's met zich meebrengen niet.

Als we een bisimulatie vast willen leggen, dan moeten we er wel voor zorgen dat we naast de verzameling toestanden ook de bewerkingen in de co-algebra's vastleggen. We geven eerst een definitie van de toestanden en de bewerking in een e-co-algebra. Nadat we hebben laten zien dat de interactie de expressie die we in e-grafen lezen niet verandert, kijken we naar o-co-algebra's.

Het lezen van een e-graaf

Wanneer we bij het lezen van een e-graaf de volgende stap zetten, dan beelden we de graaf, de kant waar we staan, en de stapel met variabelen op een symbool en een volgende toestand af.

Definitie: **de toestanden bij het lezen van de graaf in een e-interactienet** (285)

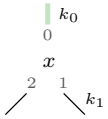
Met I_e de verzameling e-interactienetten, en S de verzameling stapels met namen van variabelen, legt

$$\{\langle \pi_1(i), k, s\rangle \mid i \in I_e \wedge k \in \pi_2(\pi_1(i)) \wedge s \in S\} \cup \{\epsilon\}$$

de verzameling T_i van toestanden vast waarin je terechtkomt wanneer je de graaf in het interactienet i leest. □

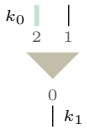
We leggen het lezen van een graaf vast als een bewerking in een co-algebra: naast het voorschrift van die bewerking (280) en de verzameling toestanden (285) hebben we voor iedere toestand waarin we bij het lezen terecht kunnen komen een passende clausule nodig.

Waaiers, variabelen die een binder zoeken, grenzen, en knopen met een ↻ als label zien we niet in de expressie terug. Wat lezen betreft slaan we ze over.



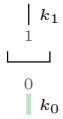
$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{286}$$

$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{287}$$

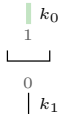


$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{288}$$

Vanwege de interactie die een paar grenzen aan de graaf toevoegt (141) hebben we in het geval van een grens twee clausules nodig.



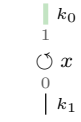
$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{289}$$



$$f_i(\langle i, k_0, (s, x)\rangle) = f_i(\langle i, k_1, s\rangle) \tag{290}$$

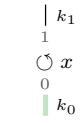
Omdat we aan een $(1, 0)$-grens niet kunnen zien of deze in een gebogen tak zit, of dat de grens een aftakking aangeeft, halen we altijd een variabele van de stapel. Dit betekent dat we bij een abstractor, onafhankelijk van het feit of deze een uitgebreid label heeft of niet, altijd een variabele op de stapel moeten leggen.

Net als een grens, kun je in twee richtingen (166) door een ↻ lopen. De eerste ↻ in een gebogen tak gaan we via de 1-poort binnen. We lezen de knoop niet, en met de stapel gebeurt er ook niets.



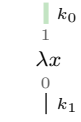
$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{291}$$

Als er nog grenzen in de gebogen tak zitten, en we bij de eerste grens de variabele van de stapel halen die we bij de abstractor op de stapel leggen, ligt aan het einde van de tak niet de juiste naam op de stapel. Daarom neemt de $\circlearrowleft$ de naam van de variabele mee, en leggen we, wanneer we door de tweede $\circlearrowleft$ in de gebogen tak lopen de variabele weer op de stapel.

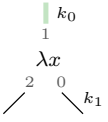


$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, (s, x)\rangle) \tag{292}$$

De abstractor zien we wel terug in de expressie. Alleen als een abstractor een gebogen tak heeft, leggen we de naam van een verse variabele op de stapel.

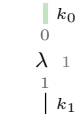


$$f_i(\langle i, k_0, s\rangle) = \langle \lambda x, \langle i, k_1, s\rangle\rangle \tag{293}$$

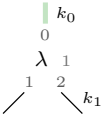


$$f_i(\langle i, k_0, s\rangle) = \langle \lambda x, \langle i, k_1, (s, x)\rangle\rangle \tag{294}$$

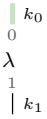
De interactie die 1 aan het label van een abstractor toevoegt verandert de oriëntatie van de knoop.



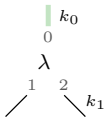
$$f_i(\langle i, k_0, s\rangle) = \langle \lambda, \langle i, k_1, s\rangle\rangle \tag{295}$$



$$f_i(\langle i, k_0, s\rangle) = \langle \lambda, \langle i, k_1, (s, x)\rangle\rangle \tag{296}$$

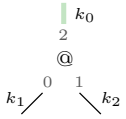


$$f_i(\langle i, k_0, s\rangle) = \langle \lambda, \langle i, k_1, s\rangle\rangle \tag{297}$$

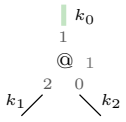


$$f_i(\langle i, k_0, s\rangle) = \langle \lambda, \langle i, k_1, (s, x)\rangle\rangle \tag{298}$$

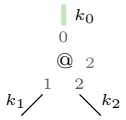
Net als bij een abstractor geeft n in het label van een applicator aan hoe vaak de interactie die applicator heeft gedraaid: een applicator begint zonder label, met één en twee keer draaien heeft de applicator 1 en 2 als uitbreiding. Bij de derde keer draaien heeft de applicator geen uitgebreid label meer.



$$f_i(\langle i, k_0, s\rangle) = \langle @, \langle i, k_1, s\rangle, \langle i, k_2, s\rangle\rangle \tag{299}$$

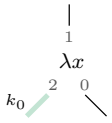


$$f_i(\langle i, k_0, s\rangle) = \langle @, \langle i, k_1, s\rangle, \langle i, k_2, s\rangle\rangle \tag{300}$$

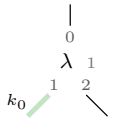


$$f_i(\langle i, k_0, s\rangle) = \langle @, \langle i, k_1, s\rangle, \langle i, k_2, s\rangle\rangle \tag{301}$$

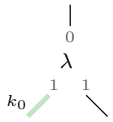
Met de richting waarin we een abstractor lezen ligt ook de poort vast waar de gebogen tak van die abstractor eindigt: volgen er nog variabelen, dan eindigt een gebogen tak in de 2-poort, anders stoppen we bij de 1-poort. De bewerking beeldt de toestand alleen op een variabele af.



$$f_i(\langle i, k_0, s\rangle) = \langle x, \epsilon\rangle \tag{302}$$



$$f_i(\langle i, k_0, (s, x)\rangle) = \langle x, \epsilon\rangle \tag{303}$$



$$f_i(\langle i, k_0, (s, x)\rangle) = \langle x, \epsilon\rangle \tag{304}$$

De lege toestand ϵ zorgt ervoor dat we niet meer in een nieuwe toestand verder gaan met lezen. Ook in het geval waarin je aan het einde van een tak een vrije variabele leest, kom je in de lege toestand terecht.

$$f_i(\langle i, k_0, s\rangle) = \langle x, \epsilon\rangle \tag{305}$$

Definitie: **de bewerking in co-algebra's bij het lezen van een e-graaf** (306)

Met de verzameling T_i van toestanden (285) en $\mathbb{S}$ de verzameling symbolen leggen de clausules (286 tot en met 305) een afbeelding

$$f_i : T_i \to \mathbb{S} \times (T_i)^{*2}$$

vast; de bewerking in de co-algebra die bij het lezen van een e-graaf hoort. □

De definitie van de bewerking (306) bevat niet voor iedere combinatie van een knoop en richting een passende clausule. We gaan daarom na of we, wanneer we stap voor de stap door de graaf lopen wel genoeg clausules hebben om altijd het einde van een tak te kunnen bereiken.

Stelling: f_i **is toereikend** (307)

Bewijs: als voor een bepaalde knoop, richting en stapel een clausule ontbreekt, dan geven we aan waarom we die clausule niet nodig hebben.

De variabele in een syntaxboom van een λ-expressie wijst met haar 0-poort in de richting van de wortel. Inspectie van de e-regels wijst uit dat de interactie in de graaf dit niet verandert, ook niet als de variabele naar haar binder zoekt. Omdat we een variabele alleen via de 0-poort binnengaan, hebben we geen andere clausules nodig.

Bij de introductie (104) wijst de waaier zonder uitgebreid label naar het einde van de tak van een abstractor. Aangezien e-interactie dit niet verandert, hoeven we niet in de richting

(0,1) of (0,2) door een dergelijke waaier te lopen. Wanneer we door een waaier lopen doen we niets met de stapel.

Alleen bij de interactie (141) die de abstractor haar eerste gebogen tak geeft, verschijnen er twee grenzen in de graaf. Inspectie van de e-regels wijst dit uit. Omdat we bij de abstractor de naam van de variabele op de stapel leggen, ligt er bij de $(1,0)$-grens dus altijd een naam op de stapel.

Met inspectie van de e-regels volgt dat we niet in de richting (0,n) door een abstractor lopen waarvan een variabele het label uitbreidt. Zodra de interactie 1 aan dit label toevoegt, volgt dat we geen regel voor de richting (2,n) nodig hebben. We gaan ook niet in de richting (1,n) wanneer tenslotte de uitbreiding verdwijnt.

De richting waarin we door een abstractor lopen, bepaalt de poort (302, 303, 303) waarin de gebogen tak van een abstractor eindigt, de poort waar we de bijbehorende variabele lezen.

Op dezelfde manier als bij de abstractor volgt dat de clausules met de applicator (299, 300, 301) alle mogelijke combinaties van poort en label weergeven: de interactie die de uitbreiding van het label van een applicator verandert, zorgt ervoor dat we de knoop steeds weer door een andere poort binnengaan. □

Omdat f_i aan de toestand waarin we ons aan het einde van een tak bevinden geen nieuwe toestand toevoegt, kunnen we de bewerking in een e-co-algebra alleen als een toereikende partiële afbeelding, en niet als een totale afbeelding opvatten.

Het lezen van een o-graaf

Wanneer we op basis van het voorschrift (280) de bewerking in de o-co-algebra vast willen leggen, hebben naast een graaf zelf, en de kant in die graaf waar we staan, ook de stapel met namen van variabelen en poorten nodig. De toestanden die bij het lezen van een o-graaf horen leggen we daarom als volgt vast.

Definitie: **de toestanden bij het lezen van de graaf in een o-interactienet** (308)

Met I_o de verzameling o-interactienetten en de verzameling stapels S legt

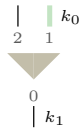
$$\{\langle \pi_1(i), k, s\rangle \mid i \in I_o \wedge k \in \pi_2(\pi_1(i)) \wedge s \in S\} \cup \{\epsilon\}$$

de verzameling T_i van toestanden vast waarin je terechtkomt wanneer je de graaf in het interactienet i leest. □

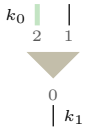
Net als bij het vastleggen van de bewerkingen in de e-co-algebra's sommen we ook in het geval van het lezen van o-grafen de nodige clausules op. We letten ook hier niet alleen op

de richting waarin je een knoop kunt lezen, want we gaan ook na of de stapel daarbij aan een bepaalde voorwaarde moet voldoen.

Omdat je een waaier die geen uitgebreid label heeft niet via de 0-poort binnen kunt gaan, hebben we alleen voor de richting (1,0) en (2,0) een clausule nodig. De stapel verandert niet, want je hoeft de naam van de poort niet te onthouden.

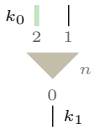


$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{309}$$



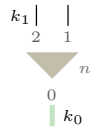
$$f_i(\langle i, k_0, s\rangle) = f_i(\langle i, k_1, s\rangle) \tag{310}$$

Als een waaier een uitgebreid label heeft, kun je er bij het lezen in drie richtingen doorheen lopen. We leggen voor ieder van die richtingen een clausule vast. Omdat er, wanneer je een poort moet kiezen, een naam op de stapel moet liggen, gelden die clausules niet voor alle mogelijke stapels.

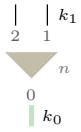


$$s_1 = (s_0, 2) \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle)$$

$$s_1 = (s_0, 1) \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{312}$$

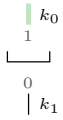


$$\triangle_-(\langle n, s_0\rangle) = \langle 2, s_1\rangle \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{313}$$

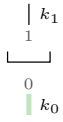


$$\triangle_{-}(\langle n, s_0\rangle) = \langle 1, s_1\rangle \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{314}$$

Een grens die geen uitgebreid label heeft kunnen we in twee richtingen passeren. Daarbij halen we namen van poorten van de stapel (210) of voegen we een $\lightning$ aan de stapel toe.

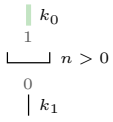


$$s_1 = \rightarrowtail(s_0) \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{315}$$

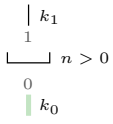


$$s_1 = (s_0, \lightning) \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{316}$$

Wanneer we een grens passeren die wel een uitgebreid label heeft, bepaalt de richting waarin we lopen het effect (242) op de stapel: we maken namen van poorten onzichtbaar of juist weer zichtbaar.

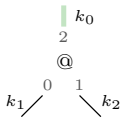


$$s_1 = \triangle_{\boxtimes}(s_0) \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{317}$$



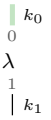
$$s_1 = \triangle_{\boxtimes}(s_0) \rightarrow f_i(\langle i, k_0, s_0\rangle) = f_i(\langle i, k_1, s_1\rangle) \tag{318}$$

Omdat geen enkele regel de oriëntatie van een applicator verandert, gaan we de knoop altijd via de 2-poort binnen.

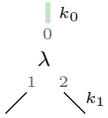


$$f_i(\langle i, k_0, s\rangle) = \langle @, \langle i, k_1, s\rangle, \langle i, k_2, s\rangle\rangle \tag{319}$$

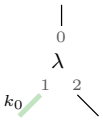
We gaan een abstractor altijd eerst via de 0-poort binnen. Alleen als de abstractor drie poorten heeft, gaan we de abstractor later, aan het einde van de gebogen tak ook via de 1-poort binnen.



$$s_1 = (s_0, x) \rightarrow f_i(\langle i, k_0, s_0 \rangle) = \langle \lambda, \langle i, k_1, s_1 \rangle \rangle \tag{320}$$



$$s_1 = (s_0, x) \rightarrow f_i(\langle i, k_0, s_0 \rangle) = \langle \lambda, \langle i, k_1, s_1 \rangle \rangle \tag{321}$$



$$s_0 = (s_1, x) \rightarrow f_i(\langle i, k_0, s_0 \rangle) = \langle x, \epsilon \rangle \tag{322}$$

Alleen aan het einde van een tak komen we een variabele tegen, vrij of gebonden. Omdat een variabele maar één poort heeft, en het label bepaalt wat we lezen, hebben we maar één clausule nodig, een clausule die geen voorwaarde stelt aan de stapel.

$$s_0 = (s_1, x) \rightarrow f_i(\langle i, k_0, s_0 \rangle) = \langle x, \epsilon \rangle \tag{323}$$

Definitie: **de bewerking in co-algebra's bij het lezen van een o-graaf** (324)

Met de verzameling T_i van toestanden (308) en $\mathbb{S}$ de verzameling symbolen leggen de clausules (309 tot en met 323) een afbeelding

$$f_i : T_i \rightarrow \mathbb{S} \times (T_i)^{*2}$$

vast; de bewerking in de co-algebra die bij het lezen van een o-graaf hoort. □

We moeten genoeg clausules hebben om een bewerking vast te kunnen leggen, dat wil zeggen: we moeten uiteindelijk altijd het einde van een tak bereiken.

Stelling: **de bewerking in een o-co-algebra is toereikend** (325)

Bewijs: we laten zien waarom we voor de richtingen, knopen en stapels waarvoor er geen clausule vastligt, deze ook niet nodig hebben.

Omdat een waaier die geen uitgebreid label heeft vanwege o-invariantie (266) altijd met haar 0-poort naar het einde van een gebogen tak van een abstractor wijst, hoeven de clausules alleen te beschrijven dat we in de richting (1,0) of (2,0) door een dergelijke waaier lopen.

Wanneer we een waaier met uitgebreid label via de 0-poort binnengaan, ligt er vanwege het evenwicht van waaiers in o-invariantie (266) altijd een naam op de stapel; clausules die een andere stapel beschrijven hebben we niet nodig. Als we in een andere richting lopen, leggen we een naam op de stapel. De clausules hoeven de stapel dan niet te beperken.

Hoewel we bij de bewerkingen $\succ\!\!\!\cdot$ en $\boxtimes$ die het effect van grenzen op de stapel vastleggen met partiële afbeeldingen (210, 242) te maken hebben, hoeven de clausules vanwege het evenwicht van abstractors en grenzen in o-invariantie (266) hier geen rekening mee te houden.

Vanwege de volledigheid (268) gaan we een applicator alleen via de 2-poort, en een abstractor alleen via de 0-poort binnen. Andere clausules hebben we niet nodig. Omdat we, wanneer we een abstractor via de 0-poort binnengaan een naam op de stapel leggen, hoeft de clausule de stapel verder niet te beschrijven.

Naderen we de 1-poort van een abstractor, dan ligt er vanwege de gebalanceerdheid van abstractors en grenzen (266) een naam van een variabele op de stapel. We hebben daarom genoeg aan een clausule die dit beschrijft. Als we aan het einde van een tak een variabele lezen, hebben we de stapel niet nodig. De clausule hoeft haar niet te beschrijven. □

Net als in de e-co-algebra kun je de bewerking alleen als een toereikende en niet als een totale afbeelding zien.

Bisimulatie en equivalente grafen

Wanneer je twee verschillende maar equivalente grafen leest, ga je via verschillende kanten van de ene naar de andere knoop. De co-algebra's die bij het lezen horen, brengen daarom ook verschillende bomen van toestanden met zich mee. We hebben nog niet laten zien dat de co-algebra's wel dezelfde bomen van symbolen met zich meebrengen.

Vanwege de definitie (50) van equivalente grafen mogen we veronderstellen dat wanneer we dergelijke grafen met elkaar vergelijken, er een afbeelding bestaat die de knopen en kanten aan elkaar toevoegt. Op basis van deze afbeelding kun je zowel op toestanden van een e-co-algebra als op die van een o-co-algebra een bisimulatie vastleggen.

Definitie: **bisimulatie en equivalente grafen** (326)

Met f_{eq} de afbeelding die de knopen in equivalente grafen aan elkaar toevoegt, en T_1 en T_2 de verzamelingen toestanden in de co-algebra's die bij het lezen van de grafen horen,

zitten $t_1 \in T_1$ en $t_2 \in T_2$ dan en slechts dan in de relatie B als

$$f_{eq}(\pi_1(\pi_2(t_1))) = \pi_1(\pi_2(t_2)) \wedge f_{eq}(\pi_2(\pi_2(t_1)) = \pi_2(\pi_2(t_2)))$$

□

Zowel de toestanden in een e-co-algebra als die in een o-co-algebra bestaan naast een graaf en een kant, ook uit een stapel; we kijken alleen naar de kanten. Als het beeld van de knopen in de kant in de eerste toestand gelijk is aan de knopen in de kant in de tweede toestand, zitten beide toestanden in de relatie.

Stelling: **bisimulatie en equivalente grafen** (327)

De relatie B is een bisimulatie.

Bewijs: vanwege de afbeelding f_{eq} die grafen tot equivalente grafen (50) maakt is het volgende symbool in beide grafen hetzelfde, en zit de eerste kant van iedere tak ook weer in B zodat de relatie aan de voorwaarde in de definitie (284) van bisimulatie voldoet. □

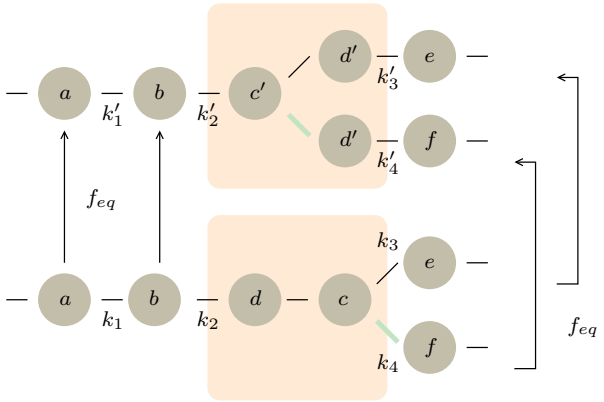
Het feit dat equivalente grafen dezelfde bomen met zich meebrengen verrast niet. Interactie zou de expressie die we in een graaf lezen kunnen veranderen. We beginnen met na te gaan hoe het wat dit betreft met e-interactie zit. Daarna kijken we naar de interactie in o-grafen.

Bisimulatie en e-interactie

Omdat we verwachten dat we ondanks de interactie in een e-graaf toch dezelfde expressie blijven lezen, proberen we een relatie op e-co-algebra's vast te leggen waarvan we kunnen laten zien dat het een bisimulatie is. Aangezien een interactiestap (58) tot equivalente partiële grafen leidt, kunnen we voor de equivalente stukken van de grafen de bisimulatie op dezelfde manier als bij equivalente grafen op basis van f_{eq} vastleggen.

De toestanden bij het lezen van de niet-equivalente delen van de grafen moeten we op een andere manier met elkaar in verband brengen. De regels die we hebben vastgelegd hebben een bepaalde vorm: als we beide knopen in het niet-equivalente fragment lezen, dan verandert er niets aan de vertakkingen.

Alleen het uitgebreide label van de knopen verandert, maar de rest van het label niet: een applicator blijft een applicator, en een abstractor blijft een abstractor. We voegen toestanden aan de relatie toe door de takken te volgen, en de knopen aan elkaar toe te wijzen zoals f_{eq} dat in equivalente stukken van de graaf doet.



(328)

Als we in beide niet-equivalente stukken maar één van de twee knopen daadwerkelijk lezen, dan zitten de toestanden waar de kanten k_2 en k_2' deel van uit maken dan en slechts dan in de relatie als de knopen die we lezen hetzelfde label hebben.

Hebben de knopen die we lezen één tak, dan maakt de eerste kant in die tak die weer in het equivalente stuk zit deel uit van de nieuwe toestand. Als een knoop die we lezen twee takken (328) heeft, dan krijgen we twee nieuwe toestanden. Zo maken k_3 en k_4 in de ene en k_3' en k_4' in de andere graaf deel uit van de nieuwe toestanden.

Omdat we in het equivalente stuk van de graaf verder gaan, geldt dat de nieuwe toestanden ook weer deel uit maken van de relatie. In het geval van een niet-equivalent fragment met een vertakking (328) zitten zowel de toestanden die bij k_3 en k_3' als de toestanden die bij k_4 en k_4' horen in de relatie.

Definitie: **bisimulatie en e-co-algebra's** (329)

Met T de verzameling van toestanden waarin we een e-graaf lezen, T' de toestanden in de co-algebra bij de graaf na een e-interactie, en met f_{eq} de afbeelding (50) die de knopen in de equivalente delen van de grafen aan elkaar toevoegt, leggen de volgende clausules een bisimulatie $B \subseteq T \times T'$ vast:

- wanneer we door de equivalente delen van de grafen (328) lopen zit $\langle t_1, t_1' \rangle$ in B dan en slechts dan als

$$\begin{aligned} f_{eq}(\pi_1(\pi_2(t_1))) &= \pi_1(\pi_2(t_1')) \quad \wedge \\ f_{eq}(\pi_2(\pi_2(t_1))) &= \pi_2(\pi_2(t_1')) \quad \wedge \\ \pi_3(t_1) &= \pi_3(t_1') \end{aligned}$$

- Als we, afgezien van het uitgebreide label, in de niet-equivalente fragmenten van de grafen dezelfde opeenvolging van symbolen lezen, dan maken de toestanden onder dezelfde voorwaarde deel uit van de relatie als in de equivalente delen van de grafen.

· Wanneer we door de niet-equivalente fragmenten van de grafen (328) lopen, en de opeenvolging van de labels van de knopen in beide fragmenten verschilt, dan zitten

$$\begin{array}{l} \langle\langle g_1, k_2, s_2\rangle, \langle g_2, k'_2, s_2\rangle\rangle, \\ \langle\langle g_1, k_3, s_3\rangle, \langle g_2, k'_3, s_3\rangle\rangle \quad \text{en} \\ \langle\langle g_1, k_4, s_4\rangle, \langle g_2, k'_4, s_4\rangle\rangle \end{array}$$

dan en slechts dan in de relatie B als

$$c = c'$$

□

Stelling: **bisimulatie en e-co-algebra's** (330)

De relatie B is een bisimulatie.

Bewijs: voor de equivalente stukken volgt op dezelfde manier als bij equivalente grafen (327) dat de relatie aan de voorwaarde (284) voldoet. Ook in het geval van de tweede clausule volgt op deze manier dat de relatie B aan de voorwaarde voldoet.

Neem aan dat t_2 en t'_2 op grond van de derde clausule in B zitten. We moeten dan laten zien dat $c = c'$ en ook dat de eerstvolgende toestanden dezelfde stapels met zich meebrengen. Een inspectie van de e-regels wijst uit dat dit bij iedere verandering van een graaf inderdaad het geval is. □

Stelling: **e-interactie is correct** (331)

Bewijs: aangezien we op de verzamelingen toestanden van de co-algebra's die bij een graaf vóór en de graaf na een e-stap horen altijd een bisimulatie vast kunnen leggen, brengen deze dezelfde boom met zich mee. De interactie in e-grafen verandert de expressie die we erin lezen niet; ze is correct. □

Bisimulatie en o-interactie

Afgezien van de interactie tussen een applicator en een abstractor, verwachten we niet dat een o-stap de expressie we lezen verandert. Wanneer we echter proberen een bisimulatie vast te leggen, observeren we dat de stapel in niet-equivalente fragmenten van de grafen soms (191, 192, 244) niet op dezelfde manier verandert.

Na de interactie van een abstractor en een waaier zonder uitgebreid label (191) ligt er een extra naam op de stapel. Omdat die bij de tweede waaier in de gebogen tak verdwijnt, en de andere waaiers in die tak elkaar in evenwicht houden, lezen we uiteindelijk toch

dezelfde variabele. Of we springen van de tak, waardoor de naam van de poort bij de grens van de stapel verdwijnt.

Heeft de waaier wel een uitgebreid label, dan verandert, omdat de interactie (192) de volgorde van de waaier en de abstractor verandert, ook de volgorde waarin we de naam van de poort en de naam van de variabele op de stapel leggen. Omdat ze echter ook de verwijzing naar de ↯ of de abstractor waar de poort bijhoort verandert, heeft de interactie geen gevolgen voor wat we lezen.

Als je bij twee grenzen zonder uitgebreid label de naam van de eerste variabele en alle namen van poorten die daar op volgen door een ↯ vervangt, dan ligt er niet hetzelfde op de stapel als na de interactie. Aangezien we dan de grenzen niet tegenkomen, blijven de namen die we vóór de interactie weghalen op de stapel liggen.

Omdat de waaiers in de tak tussen $(0,1)$-grens en een $(1,0)$-grens elkaar in evenwicht houden, maakt het echter niet uit of er, wanneer we in een dergelijke tak terechtkomen, iets op de stapel ligt of niet. We halen alleen de namen van poorten van de stapel die we erop leggen, of er al op lagen, en uiteindelijk komen we in de $(1,0)$-grens waar we, net als vóór de interactie, het nodige van de stapel halen.

Met de interactie van een abstractor en een waaier, en de interactie van twee grenzen zonder uitgebreid label, volgt dat de stapels in overeenkomstige toestanden van de o-co-algebra's hetzelfde zijn. We leggen daarom een bisimulatie vast op toestanden met equivalente stapels.

Definitie: **equivalente stapels** (332)

Met

S de verzameling (209) stapels,

$s_1, s_2, s_3, s_p \in S$,

$\cdot$ een bewerking die stapels samenvoegt,

s_p een stapel met alleen namen van poorten,

x een verse variabele,

$\langle (s_1, x) \cdot s_p, (s_1, \lightning) \rangle \in R$,

$\langle ((s_2, p), x) \cdot s_3, ((s_2, x), p) \cdot s_3 \rangle \in R$,

vormt de R-afsluiting van S de verzameling S_e van equivalente stapels. □

Het eerste paar hoort bij het verschil van de stapel vóór en na de interactie van twee grenzen zonder uitgebreid label. Het tweede paar hoort bij het verschil dat de interactie van een abstractor en een waaier met een uitgebreid label teweegbrengt.

Definitie: **bisimulatie en o-co-algebra's** (333)

Met T, en T' verzamelingen van toestanden in o-co-algebra's, zitten $t \in T$ en $t' \in T'$ dan en slechts dan in een relatie $B \subset T \times T'$ als

- de kanten in t en t' aan dezelfde voorwaarde voldoen als de kanten in de bisimulatie (329) bij e-co-algebra's, en als
- de stapels in t en t' equivalent zijn.

□

Stelling: **bisimulatie en o-co-algebra's** (334)

De relatie B is een bisimulatie.

Bewijs: volg de redenering in het bewijs van de stelling (330) over e-co-algebra's. Neem echter wel de bewerking (324) de interactieregels (263) en de bisimulatie (333) die bij o-co-algebra's horen.

Vóór en na de interactie van een abstractor en een waaier (191, 192) verandert de stapel niet op dezelfde manier, maar de stapels zijn wel equivalent. Vanwege het evenwicht van waaiers in o-invariantie (266) kiezen we daarom in iedere waaier de juiste poort. Omdat we bij de abstractor wel dezelfde naam op de stapel leggen, lezen we aan het einde van de tak dezelfde variabele.

Wanneer een grens op een abstractor inwerkt (206, 207) verandert alleen de zichtbaarheid van de namen die voorafgaan aan de naam van de variabele die we bij de abstractor op de stapel leggen. De opeenvolging van de namen die we in de tak op de stapel leggen verandert niet.

Heffen twee grenzen die geen uitgebreid label hebben (244) elkaar op, dan verandert de opeenvolging wel. Omdat de stapels echter equivalent zijn, volgt met het evenwicht van waaiers in een gebogen tak dat we in een gebogen tak de juiste poorten kiezen, en na de grens die bij de tweede grens hoort de graaf weer met dezelfde stapel lezen. □

We kijken tenslotte naar de interactie van een applicator en een abstractor. Omdat bij die interactie beide knopen verdwijnen, en je alleen nog de tak van de abstractor leest, en in het geval van een abstractor met een gebogen tak daarna ook nog de 1-tak van de applicator, heeft het geen zin zin te proberen een bisimulatie voor de hele graaf vast te leggen: je leest een boom met een andere structuur.

Stelling: **de interactie van een applicator en een abstractor is correct** (335)

Bewijs: hoewel het niet lukt de toestanden vóór en na de interactie via een bisimulatie met elkaar te verbinden, kun je, als je alleen door equivalente fragmenten van de graaf loopt, de tak van de abstractor vóór de interactie wel in verband brengen met de tak na de interactie. Hetzelfde geldt voor de 1-tak van de applicator.

Als je vanaf het begin van de tak van de abstractor alleen nog door equivalente stukken loopt, dan komt de interactie met substitutie overeen: vanwege het evenwicht lezen we aan het einde van de gebogen tak van de abstractor namelijk hetzelfde in de 1-tak als vóór de interactie.

Ook als de abstractor geen gebogen tak heeft komt de interactie met substitutie overeen, want in dat geval lezen we namelijk alleen de tak van de abstractor.

Als je wel door het niet-equivalente fragment loopt, volgt de correctheid van de interactie niet onmiddellijk. Omdat je echter maar eindig vaak door dit fragment kunt lopen, komt de laatste verandering wel met substitutie overeen, zodat de correctheid van de voorafgaande veranderingen volgt. □

Als je wel door het niet-equivalente fragment loopt, volgt de correctheid van de interactie niet onmiddellijk. Omdat je echter maar eindig vaak (253) door dit fragment kunt lopen, komt de laatste verandering wel met substitutie overeen, zodat de correctheid van de voorafgaande veranderingen volgt.

Stelling: **o-interactie is correct** (336)

Bewijs: met de correctheid van de interactie van een applicator en een abstractor, en het feit dat je op de co-algebra's bij de grafen die bij de rest van de interactie horen altijd een bisimulatie vast kunt leggen volgt dat o-interactie correct is. □

Omdat de verandering die de interactie van een applicator en een abstractor met zich meebrengt op een bepaalde manier met de substitutie in de λ-rekenkunde overeenkomt, en de rest van de o-interactie de expressie die je leest niet verandert, is de interactie correct: ze doet niets wat je in die rekenkunde niet zou kunnen doen.

De interactie in een o-graaf behoudt ook de binding (170) die e-interactie bewerkstelligt.

Stelling: **lokale binding** (337)

De binding van variabelen in een o-graaf is lokaal.

Bewijs: in een e-graaf is de binding van variabelen lokaal. Wanneer we de regels voor o-interactie inspecteren, dan volgt dat als we vóór en na de interactie een variabele lezen, we dit in dezelfde knoop doen: de binding van variabelen blijft lokaal. □

Optimale stappen

Met bisimulatie, een raamwerk waarin we vergelijken wat we in twee grafen lezen, volgt dat de interactie correct is. Omdat de interactie ook volledig is, kan ze de stappen die we in de λ-rekenkunde zouden zetten volgen. Maar het correcte en volledige alleen impliceren niet dat we bij o-interactie ook optimale stappen zetten. We laten nog zien dat dit, ook weer op basis van de invariant, echter wel volgt.

7 Alles bij elkaar

De interactie die we in onderdeel 5 vastleggen kan de stappen in de reductie van een expressie in de λ-rekenkunde volgen. We moeten nog laten zien dat we de interactie ook als een algoritme voor de optimale reductie van λ-expressies op mogen vatten. Dit doen we door het algoritme met dat van Asperti en Guerrini te vergelijken.

We zetten de eigenschappen van de interactie die syntaxbomen in grafen vertaalt, en de eigenschappen van de interactie die 'optimale' stappen zet op een rij. Daarbij zien we dat het evenwicht van de grenzen en waaiers, en het gebruik van bisimulatie het gemakkelijker maken de correctheid van de interactie te bewijzen.

Op basis van de eigenschappen overwegen we of de interactie een plaats in kan nemen in de praktijk. We gaan bovendien na of we het oorspronkelijke doel hebben bereikt, met andere woorden, of we de interactie kunnen beschrijven zonder dat we het 'orakel' een betekenis moeten geven die buiten de graaf zelf ligt.

Omdat we bomen die de syntax van een λ-expressie weergeven vanwege interactie die optimale stappen zet in o-grafen vertalen, en die grafen wel een λ-expressie, maar niet de syntax van die expressie weergeven, voorzien we tenslotte ook in regels voor de interactie die dergelijke grafen weer in bomen verandert.

Interactie

De interactie (174) die we in onderdeel 3 vastleggen, e-interactie, verandert de boom die de syntax van een λ-expressie weergeeft in een graaf. De interactie die volgt kan deze graaf zodanig herschrijven dat de verandering van de expressie die we in de opeenvolgende grafen lezen met optimale stappen overeenkomt. We zeggen daarom ook wel dat die interactie optimale stappen zet.

Bij de interactie (265) die op de e-interactie volgt, o-interactie, hebben we met twee soorten stappen te maken. De interactie zet stappen, zodanig dat de verandering in de expressie die we in de graaf lezen met een optimale stap overeenkomt. Daarnaast maakt ze met andere stappen de 'optimale' stappen mogelijk. We beschrijven de o-regels in onderdeel 4 en in onderdeel 5.

De interactie mag grafen niet op een zodanige manier veranderen dat we er een expressie in zouden lezen die we niet krijgen wanneer we de expressie die we in de syntaxboom lezen in de λ-rekenkunde zouden reduceren. In onderdeel 6 laten we zien dat we de interactie daadwerkelijk als correct (331, 336) op mogen vatten.

De stappen moeten niet alleen in orde zijn. Omdat we de expressie die de reductie in de λ-rekenkunde oplevert, uiteindelijk ook in de graaf willen lezen, moet ze ook alle nodige

stappen zetten. We laten daarom zien dat de interactienetten over alle nodige regels (175, 268) beschikken.

Als we een e-normaalvorm als een o-graaf interpreteren, voorziet e-interactie gevolgd door o-interactie in een algoritme dat met het herschrijven van grafen de optimale reductie van een expressie weergeeft. Dit algoritme heeft genoeg aan de boom die de syntax van de λ-expressie weergeeft die we willen reduceren.

Omdat het eo-algoritme de vorm heeft van interactieregels, verandert het grafen niet alleen op een lokale (42) maar ook op een parallelle (45) en asynchrone (46) manier. Het lokale maakt de interactieregels overzichtelijk. Ook Asperti en Guerrini leggen interactienetten vast. Lamping doet dit nog niet, hij voegt ook niet lokale regels aan zijn algoritme toe.

Correctheid op basis van evenwicht

Wanneer we de correctheid van de interactie van knopen in grafen bewijzen, maken we gebruik van bisimulatie. Hoewel we in onderdeel 6 zien dat bisimulatie zelf een redelijk ingewikkelde beschrijving met zich meebrengt, blijft een bewijs op basis van bisimulatie (330, 334) beperkt. De definitie van een relatie en inspectie van lokale veranderingen, de regels voor de interactie, volstaat.

In een redenering die betrekking heeft op een regel hebben we alleen de invariant (166, 251) nodig. Deze brengt het evenwicht van grenzen (199, 250) en waaiers (241) in de gebogen takken van abstractors met zich mee. Het evenwicht is een eigenschap van de takken in een graaf, niet van de opeenvolging van knopen die we krijgen wanneer we een graaf lezen.

Bij de vertaling van een expressie in een graaf [1, 3.3] voegen Asperti en Guerrini haakjes en grenzen toe. Maar niet op een zodanige manier dat de knopen elkaar in de gebogen takken van een abstractor in evenwicht houden. Daardoor raken waaiers, zelfs die welke aan het begin en het einde van een dergelijke tak verschijnen, hun evenwicht kwijt.

Vanwege het gebrek aan evenwicht van de knopen in de takken krijgt het bewijs van de correctheid van het AG-algoritme een minder lokaal karakter. De redenering in het bewijs gaat namelijk ook over paden, dat wil zeggen, opeenvolgingen van knopen die we krijgen wanneer we een graaf lezen.

Wanneer we een graaf lezen hebben we een stapel (209) nodig. Hoe de context waarin we grafen lezen tot stand komt beschrijven we in onderdeel 4. Omdat we bij een bewijs van de correctheid van interactie in grafen lezen, hebben we in een dergelijk bewijs ook een stapel nodig. We mogen het daarom niet als syntactisch opvatten.

De context waarin we een o-graaf lezen heeft een eenvoudigere structuur dan de context die we nodig hebben wanneer we een AG-graaf lezen. Bij een o-graaf leggen we namelijk

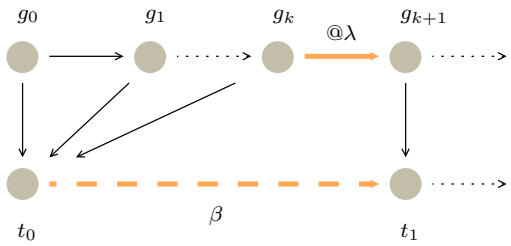
alleen namen van variabelen en poorten, en de nodige aanduidingen op de stapel. De stapel bij een AG-graaf bevat zelf ook weer stapels.

De vorm die we aan de context geven heeft echter geen wezenlijke gevolgen voor de manier waarop een algoritme werkt, want we hebben de context alleen dan nodig als we lezen. Het wel of niet in evenwicht zijn van de één of andere knoop zien we niet als een gebrek van het algoritme.

Voor wat betreft het syntactische maakt het geen verschil wanneer we, zoals we dat dadelijk doen, interactie vastleggen die een o-graaf weer in een boom verandert die de syntax van een expressie weergeeft. Zelfs als we die interactie vastleggen, moeten we, wanneer we zouden willen laten zien dat ook die interactie correct is, grafen met een stapel lezen.

Optimale stappen

Als we de syntaxboom van een λ-expressie als e-graaf zien, dan verandert e-interactie deze graaf in een graaf waarin we met o-interactie optimale stappen kunnen zetten. Vanwege het correcte en het volledige van de interactie lezen we in de o-graaf uiteindelijk de expressie die we krijgen wanneer we de oorspronkelijke expressie in de λ-rekenkunde reduceren.



(338)

Omdat er in de 0-tak van een applicator na een abstractor een waaier kan volgen, en we daardoor vaker door de applicator en de abstractor kunnen lopen, hoort er bij die applicator mogelijk meer dan één reduceerbare applicatie. Hoewel de interactie tussen de abstractor en de applicator meer dan één applicatie weg kan werken, moeten we nog laten zien dat het met die interactie om een optimale stap gaat.

De interactie in o-grafen moet de lokale binding die e-interactie bewerkstelligt (170) in stand houden. Zou er geen sprake meer zijn van lokale binding, dan moeten we de tak van de applicator met meerdere knopen in de graaf verbinden. De interactie tussen een applicator en een abstractor zouden we dan hoe dan ook niet meer als een optimale stap op kunnen vatten.

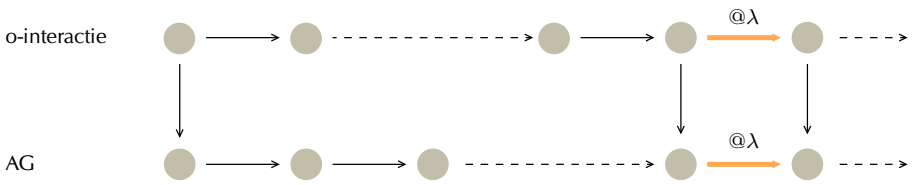
Met alleen lokale binding volgt echter niet dat we met de interactie van een applicator en een abstractor een optimale stap zetten. We moeten de graaf en meer in het bijzonder de

interactie van een applicator en een abstractor in verband brengen met dat wat Lévy als optimaal opvat.

Asperti en Guerrini laten zien [1, p.129] dat als een applicator en een abstractor op elkaar inwerken, dit voor wat de expressie die we lezen overeenkomt met het wegwerken van een verzameling applicaties die in dezelfde familie zitten. Daarbij veronderstellen ze [1, definitie 5.5.1] lokale binding.

Het volledige van het AG-algoritme houdt in dat de interactie tussen een applicator en een abstractor alle applicaties in de expressie wegwerkt die in dezelfde familie zitten. De manier waarop Asperti en Guerrini het volledige opvatten verschilt dus van het volledige van e-interactie en o-interactie.

Omdat we met andere grafen, andere knopen en andere regels voor de interactie te maken hebben, verschillen de stappen die in een o-interactienet de interactie van een applicator en een abstractor mogelijk maken van de interactie die dit in een AG-graaf doet.



(339)

Maar vanwege het correcte, volledige, en lokale binding (337) kunnen we de redenering in het bewijs van Asperti en Guerrini ook op o-interactie toe toepassen. Hiermee volgt dat met de interactie van een applicator en een abstractor ook bij o-interactie alle applicaties verdwijnen die in dezelfde familie zitten.

Als de interactie van een applicator en een abstractor alleen applicaties wegwerkt die in dezelfde familie zitten, wil dit nog niet zeggen dat die interactie met een optimale stap overeenkomt. We moeten namelijk een mechanisme aan het eo-algoritme toevoegen dat er voor zorgt dat het geen onnodige stappen zet. Asperti en Guerrini geven een suggestie [1, 5.6] voor een dergelijk mechanisme.

De effciëntie van o-interactie

Hoewel we met o-interactie en interactie die de regels in het AG-algoritme volgt dezelfde optimale stappen zetten, verschillen de algoritmen wat de rest van de stappen betreft. Zo brengt het evenwicht van waaiers bij o-interactie een eenvoudig bewijs van de correctheid van die interactie met zich mee.

We vergelijken beide algoritmen ook door het aantal stappen te tellen die volgen in een graaf waarin we de dezelfde λ-expressie lezen. Die expressie geven we weer in de eerste kolom. De andere kolommen bevatten het aantal stappen in de interactie ingedeeld naar de soort. Met 'waaier' gaat het om de interactie tussen twee waaiers.

Asperti en Guerrini implementeren hun algoritme in de vorm van wat zij de 'Bologna Optimal Higher-order Machine' [2] noemen. Aangezien de regels voor o-interactie en de regels die de beschrijving [16] van Lambdascope weergeeft tot dezelfde interactie leiden, laten we o-interactie door een implementatie van dat algoritme vertegenwoordigen.

De stappen waarmee we een verzameling applicaties wegwerken die in dezelfde familie zitten noemen we een familiestap. Naast dergelijke stappen en stappen waarin waaiers op elkaar inwerken, geven we het aantal overige stappen weer. Wat het AG-algoritme betreft gaat het daarmee om interactie met haakjes en croissants, bij Lambdascope gaat het over interactie met grenzen.

f	BOHM 0.0			Lambdascope		
$\lambda n.(n)\underline{\underline{2}}2II$	waaier	familiestap	overig	waaier	familiestap	overig
$f\underline{1}$	42	12	228	13	12	142
$f\underline{2}$	67	17	681	34	17	270
$f\underline{3}$	100	22	6879	71	22	551
$f\underline{4}$				140	27	1299
$f\underline{5}$				273	32	3646

Wanneer we de algoritmen op dezelfde λ-expressie toepassen, leidt dat tot hetzelfde aantal familiestappen. We verwachten ook niets anders. Alleen als een algoritme niet met een strategie werkt die onnodige stappen uit de weg gaat, kan er een verschil in het aantal familiestappen ontstaan.

Als we naar het aantal stappen met waaiers, en naar het aantal overige stappen kijken, mogen we stellen dat Lambdascope efficiënter is dan BOHM 0.0, de implementatie van het AG-algoritme.

Omdat haakjes en croissants in een gebogen tak van een abstractor elkaar niet in evenwicht houden, en de interactie van deze knopen met een waaier de uitbreiding van het label van die waaier verandert, houden de waaiers in de takken elkaar niet meer in evenwicht. Dit draagt bij aan een verklaring van het feit dat het AG-algoritme het wat waaiers betreft minder goed doet dan Lambdascope.

Zelf geven Asperti en Guerrini aan [1, 9.1] dat het algoritme dat zij beschrijven last heeft van opeenhopingen van haakjes en croissants. Hierdoor blijven er meer van deze knopen over, en dit leidt, wanneer ze tussen een applicator en een abstractor in zitten, tot extra interactie. Dit verklaart waarom BOHM het ook wat de overige interactie betreft minder goed doet dan Lambdascope.

Door extra regels [1, 9.3] aan hun algoritme toe te voegen beperken Asperti en Guerrini de opeenhopingen. We vergelijken de nieuwe versie van BOHM met Lambdascope.

f	BOHM 1.1			Lambdascope		
$\lambda n.(n)\underline{22}II$	waaier	familiestap	overig	waaier	familiestap	overig
$f\underline{1}$	22	12	16	13	12	142
$f\underline{2}$	44	17	33	34	17	270
$f\underline{3}$	74	22	74	71	22	551
$f\underline{4}$	120	27	199	140	27	1299
$f\underline{5}$	198	32	636	273	32	3646
$f\underline{6}$	340	37	2273	534	37	11872
$f\underline{7}$				1051	42	42601
$f\underline{8}$				2080	47	161337
$f\underline{9}$				4133	52	628112
$f\underline{10}$	4236	57	527413	8234	57	2479086

Vanaf $f\underline{4}$ doet BOHM 1.1 het duidelijk beter dan Lambdascope. De verbetering heeft zowel effect op de interactie tussen waaiers als de overige interactie.

Omdat de nieuwe regels voor de interactie opeenhopingen beperken, zitten er in de grafen waar het algoritme mee werkt ook minder haakjes en croissants de interactie van een applicator en een abstractor in de weg. Dit heeft als gevolg dat er ook minder interactie met dit soort knopen volgt. Dit verklaart het kleinere aantal overige stappen.

Minder haakjes en croissants betekent ook dat de waaiers in de takken elkaar vaker in evenwicht houden, elkaar daardoor ook vaker opheffen, zodat waaiers elkaar minder vaak tegenkomen. Dit geeft aan waarom het algoritme met de nieuwe regels het ook wat waaiers betreft beter doet dan het oorspronkelijke.

Het ligt minder voor de hand waarom het algoritme met de nieuwe regels minder stappen met waaiers nodig heeft dan het wat die knopen betreft evenwichtige Lambdascope. Want de regels voor de interactie verschillen niet wezenlijk. De uitbreidingen van de labels van

waaiers in de vertalingen van bomen verschillen echter wel.

Vanwege het verschil in uitbreidingen, en de beperking van de opeenhopingen zou een waaier die naar het einde van een gebogen tak wijst de tweede waaier in een paar dat in die tak verschijnt op kunnen heffen, dit terwijl o-interactie dat niet kan. Hierdoor zouden er uiteindelijk minder waaiers overblijven.

Terwijl BOHM er vanwege het beperken van de opeenhopingen van haakjes en croissants voor wat betreft het aantal stappen een gunstiger beeld laat zien dan de implementatie van Lambdascope, neemt het aantal haakjes dat dit algoritme vanwege de regel voor de interactie van de applicator en de abstractor toevoegt alleen maar toe.

Zoals Liu en Hudak opmerken op [7] moeten we naast optimale stappen ook rekening houden met de stappen die de optimale stappen mogelijk maken. We zouden daarom na moeten gaan of we de interactie waar waaiers en andere knopen aan meedoen kunnen beperken.

De regels die voor BOHM de opeenhoping van haakjes en croissants beperken, zorgen voor een opmerkelijke verbetering van dit algoritme. Omdat Lambdascope geen last heeft van opeenhopingen [16, 8.5] volgt dat we dit algoritme niet op een vergelijkbare manier kunnen verbeteren.

Als we met een algoritme dat optimale stappen zet λ-expressies willen reduceren, dan moeten we vanwege de efficiëntie BOHM boven Lambdascope kiezen. We zouden ook kunnen proberen één van beide algoritmen te verbeteren. Mackie [13] laat het idee van optimale stappen helemaal los, en richt zich op het verbeteren van de efficiëntie.

Uitbreiding van de praktijk

De manier waarop we λ-expressies weergeven, interpreteren en veranderen bepaalt de praktijk waarin we met deze expressies werken. We interpreteren grafen aan de hand van een stapel, en interactienetten bepalen de verandering van grafen. We gaan na op welke manier het gebruik van grafen de praktijk uitbreidt.

We hebben laten zien dat e-interactie en o-interactie correct en volledig is. Daarnaast hebben we vastgesteld dat we met o-interactie optimale stappen zetten. Hoewel we de algoritmen op meer expressies toe zouden willen passen, mogen we Lambdascope redelijk efficiënt noemen. Op grond van dit alles vindt e-interactie en o-interactie een plaats in de praktijk van het rekenen met λ-expressies.

Nog afgezien van de definitie van bepaalde regels in het bijzonder, kunnen we interactie op zich als een uitbreiding van de praktijk zien. Omdat we ook de substitutie in expressies in de De Bruijn rekenkunde (80) de vorm van interactie kunnen geven, hebben we wel met een neutrale uitbreiding te maken.

Als gevolg van het feit dat we geen dubbel werk willen doen, moeten we op de één of andere manier uitdrukking geven aan het gemeenschappelijke. Omdat een boom die de syntax van een expressie weergeeft dit niet kan, hebben we met een weergave in de vorm van waaiers van het gemeenschappelijke met een wezenlijke uitbreiding van de praktijk te maken.

Bij de interpretatie van grafen hebben we ook een stapel met namen nodig. Ook die komen we in andere onderdelen van de praktijk tegen. Kijken we bijvoorbeeld naar expressies in de De Bruijn rekenkunde, dan hebben we tenminste een eenvoudige stapel met namen van variabelen nodig om een boom die de syntax van een expressie weergeeft te interpreteren.

We leggen naast namen van variabelen ook namen van een poort die we in een waaier moeten kiezen op de stapel. Hierin verschilt de manier waarop we de stapel gebruiken met de stapel die we nodig hebben bij het interpreteren van expressies in de De Bruijn rekenkunde. Vanwege waaiers en grenzen moeten we ook zoeken in de stapel. Maar in de stapel waarmee we De Bruijn expressies interpreteren doen we dat ook.

Alles bij elkaar wijken we met de stapel die we nodig hebben bij het interpreteren van grafen nauwelijks af van wat we in de praktijk van het rekenen met λ-expressies doen.

Duurzame betekenis

Om ervoor te zorgen dat we aan het einde van een tak de juiste variabele lezen, en in een waaier de juiste poort kiezen, hebben we knopen nodig die een effect op de stapel hebben. We voegen daarom waaiers en grenzen aan grafen toe, en brengen deze knopen via de stapel met elkaar in verband.

We betrekken grenzen via de stapel ook op andere objecten in de praktijk. Zo heeft de bewerking (p.116) die Bird en Paterson vastleggen een vergelijkbaar effect op de stapel die we nodig hebben wanneer we een De Bruijn expressie interpreteren. De grenzen breiden de praktijk dus niet op een buitensporige manier uit.

We kunnen de grenzen in een graaf ook in verband brengen met een expressie waarin λ het einde van het bereik van een abstractor aangeeft. Maar terwijl met het verdwijnen van die abstractor ook het bereik verdwijnt, blijven de grenzen in de graaf achter. We kunnen grenzen daardoor alleen tijdelijk met het bereik van een abstractor in verband brengen.

De grens in een graaf gaat vanwege interactie nog op een andere manier verschillen. Zodra de grens een uitgebreid label krijgt, heeft deze een ander effect op de stapel (p.127) dan een bewerking die het einde van het bereik van een abstractor aangeeft. Bij die bewerking verdwijnt er definitief een naam van de stapel, bij een grens met een uitgebreid label niet.

Hoewel een grens in een graaf oorspronkelijk het einde van een bereik van een abstractor in een λ-expressie weergeeft, lukt het, omdat het bereik verdwijnt en de grenzen blijven,

niet de grens op een duurzame manier met de opvatting het bereik van een abstractor te verbinden.

Het lijkt erop dat de bewerking die Bird en Paterson voorstellen wel een definitief karakter heeft. De betekenis die we er in een expressie aan toe kennen is onafhankelijk van wat er verder met de graaf gebeurt. De grens in een graaf heeft ook een definitieve betekenis, maar alleen vanwege de stapel en de waaiers in de graaf.

De ↯ in een De Bruijn expressie (p.118) en de grenzen die de interactie van de applicator en de abstractor toevoegt hebben een vergelijkbaar effect op de stapels. Omdat de ↯ en een grens dezelfde rol spelen, zouden we de grens op een ↯ kunnen betrekken. Maar een dergelijke betekenis heeft niets meer met het bereik van een abstractor te maken.

Omdat de correctheid van o-interactie grafen met expressie in verband brengt, hebben grenzen daarmee wel een duurzame betekenis. De grenzen hebben ook een betekenis op het vlak van grafen. Ze vormen een mechanisme waarmee we, wanneer we een graaf interpreteren in een waaier de juiste poort kunnen kiezen.

Het orakel

Asperti en Guerrini [1, p.303] hebben het over het definitieve doorgronden van het 'orakel' in het algoritme van Lamping. Ze doelen daarmee op het minder doorzichtige in het mechanisme dat bij de interpretatie van grafen waar dat algoritme mee werkt ervoor moet zorgen dat we in een waaier de juiste poort kiezen.

De opmerking [1, p.39] over het feit dat Lamping's algoritme alleen werkt als we extra haakjes toevoegen, maakt dit algoritme geheimzinnig. Zelf voegen Asperti en Guerrini echter ook knopen toe die ervoor zorgen dat we in waaiers de juiste poort kiezen. Ze geven echter een uitgebreidere verklaring dan Lamping.

Wanneer we de extra knopen proberen te verklaren breken we het geheimzinnige af. Zo geven de haakjes in Lambdascope na het vertalen van een λ-expressie in een graaf waar dat algoritme mee werkt het bereik van een abstractor weer. Maar alleen dan. Want door interactie geven de haakjes niet langer het bereik weer.

Het meer of minder doorzichtige bij het interpreteren van een graaf staat het correcte, en daarmee de plaats van een algoritme in de praktijk niet in de weg. Maar een echt orakel zou de praktijk onmogelijk maken. Want vanwege het correcte moeten we wel weten hoe de interpretatie van een graaf tot stand komt.

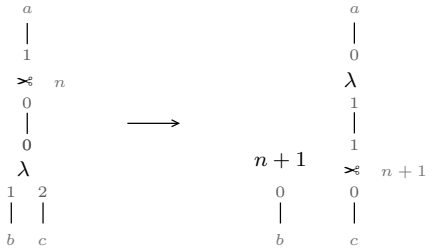
Wanneer we de beschrijving van de regels voor o-interactie bekijken, dan zien we dat we daarin de knopen geen betekenis geven die buiten de graaf ligt. We hoeven het 'orakel' daarom ook niet verder te verklaren dan dat we het nodig hebben voor de correcte werking van de interactie. Het is in dit opzicht dat we een andere benadering (p.32) volgen.

Weer terug

Alleen vanwege het bewijs van correctheid moeten we o-grafen interpreteren. Na het toepassen van het eo-algoritme kan verdere interactie de uiteindelijke o-graaf namelijk weer in een boom veranderen. Ook op die manier kunnen we achterhalen welke expressie de o-graaf representeert.

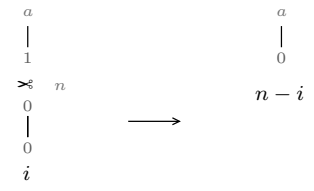
We bekijken regels voor de interactie die een o-graaf in een boom verandert die de syntax van een De Bruijn expressie weergeeft. Omdat we, na de interactie die takken buigt en vastmaakt, en interactie die optimale stappen zet, weer teruggaan naar een boom die een expressie weergeeft, noemen we de interactie t-interactie.

Omdat er een boom geen gebogen takken heeft, voegen we bij de wortel een knoop aan de graaf toe die de gebogen takken los knipt. Knipt de knoop de takken van een abstractor los, dan voegt ze aan het einde van die takken een nieuwe knoop toe, een knoop die we na de interactie als een De Bruijn index op mogen vatten.



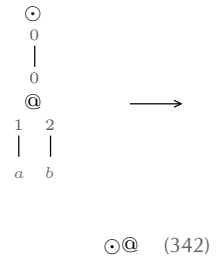
(340)

We laten de nieuwe knoop het aantal abstractors tellen dat ze, op weg naar het einde van een tak waar ze in zit tegenkomt. Wanneer de ⋊-knoop het einde van de gebogen takken bereikt, is het verschil van de uitbreidingen in de labels van de ⋊-knoop en de knoop die aan het einde zit gelijk aan de De Bruin index van de variabele die we aan het einde van de tak lezen.

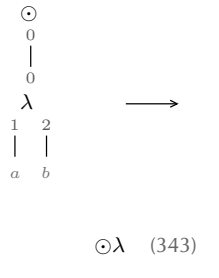


(341)

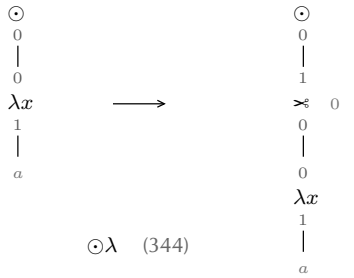
Wanneer een ⋊-knoop een applicator tegenkomt, gaat er zowel in de ene als de andere tak een ⋊-knoop verder. De ⋊-knopen bewegen zich naar het einde van de takken door over de andere knopen in de graaf te springen.



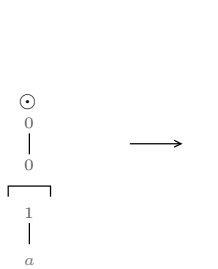
⊙@ (342)



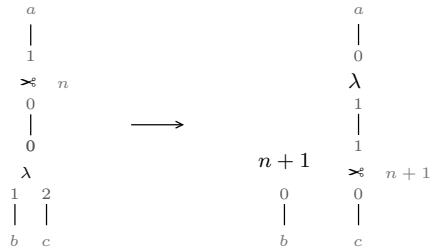
⊙λ (343)



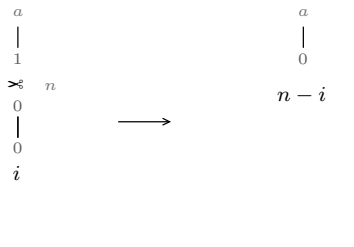
⊙λ (344)



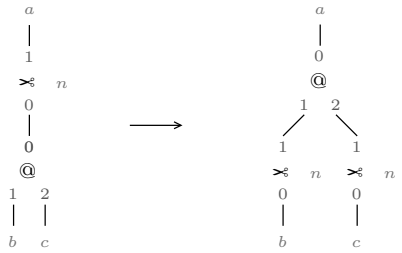
⊙− (345)



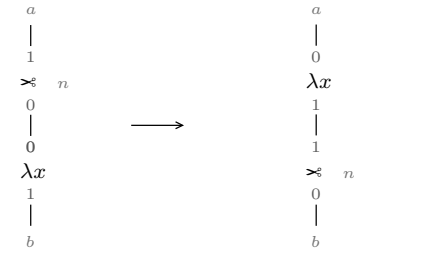
≻₀λ (346)



≻₀i (347)



≻₀@ (348)



≻₀λ (349)

≻₀x (350)

Omdat een applicator in een o-graaf niet met haar 0-poort aan de wortel vast kan zitten, zou de $\succ_\circ$-knoop niet in de graaf kunnen verschijnen als er onmiddellijk na de wortel een applicator volgt. Applicators moeten daarom eerst draaien. We laten de interactie dit op dezelfde manier doen als e-interactie, de interactie die de takken vastmaakt, dit doet.

Vanwege de volledigheid van o-interactie kan de 0-tak van o-graaf niet met een applicator beginnen, en daardoor kan een abstractor er niet voor zorgen dat een applicator draait. De tak van een applicator kan wel met een applicator beginnen. Maar die applicator wijst nog niet haar 0-poort naar de wortel van de graaf.

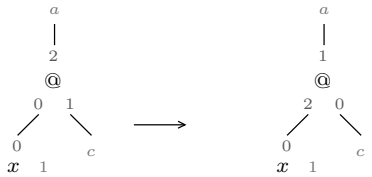
Als er geen abstractor voorafgaat, moet de tak in een vrije variabele eindigen. Die kan de applicator laten draaien, en deze applicator draait dan op haar beurt de voorafgaande applicators. Als er wel een abstractor voorafgaat, dan knipt de $\succ_\circ$-knoop de gebogen takken van die abstractor los, en kan de index de applicator laten draaien.

De 1-tak van een applicator in een o-graaf kan wel met een abstractor beginnen. Als interactie een applicator één keer heeft gedraaid, kan een abstractor die applicator nog een keer draaien. Hier ziet ernaar uit dat we, wat het draaien van applicators betreft, genoeg hebben aan de eerste vijf regels van het tweede (p.211) overzicht.

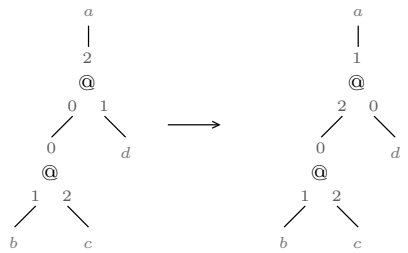
Met het draaien van de applicator kan de $\succ_\circ$-knoop verder. De $\succ_\circ$-knoop werkt alleen op een $(0,1)$-grens zonder uitgebreid label in. Omdat we in de boom die de syntax van een λ-expressie weergeeft geen grenzen nodig hebben, haalt de interactie de grens uit de graaf. Het uitgebreide label van de $\succ_\circ$-knoop verandert daarbij niet.

(351)

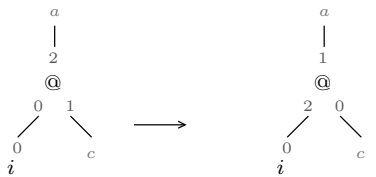
De $\succ_\circ$-knoop duwt de rest van de knopen voor zich uit. Omdat de interactie van een applicator en een abstractor in een o-graaf ook tot het verplaatsen van knopen leidt, voegen we de regels voor de interactie in die grafen aan de verzameling t-regels toe. Vanwege het draaien van de applicator hebben we echter wel een andere regel (355) voor de interactie van een applicator en een abstractor.



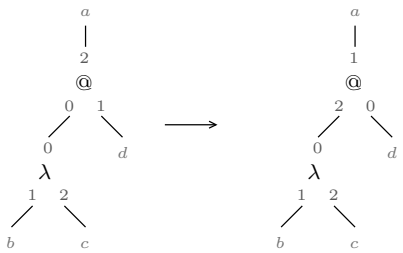
$x@$ (352)



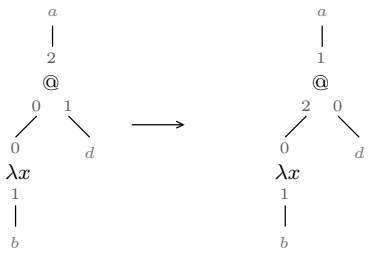
$@@$ (353)



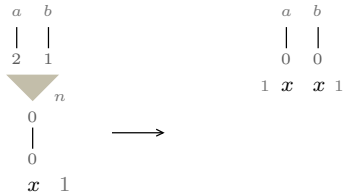
$i@$ (354)



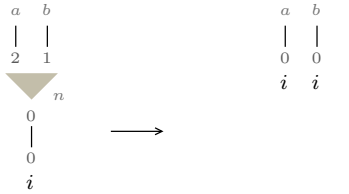
$@\lambda$ (355)



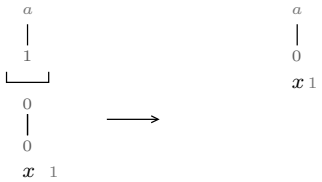
$@\lambda$ (356)



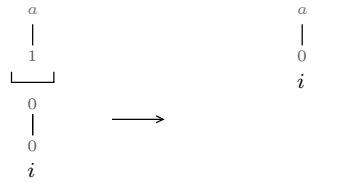
$\triangle x$ (357)



$\triangle i$ (358)



$-x$ (359)



$-i$ (360)

De waaiers en grenzen bewegen zich naar het uiteinde van de takken. Naast de regels (p.211) voor waaiers en grenzen met een uitgebreid label hebben we ook overeenkomstige regels nodig voor waaiers en grenzen die geen uitgebreid label hebben.

Omdat e-interactie het label van vrije variabelen uitbreidt, hebben deze variabelen in een t-graaf nog steeds een uitgebreid label. Wanneer een $\succ\!\!<$-knoop aan het einde van een tak een vrije variabele tegenkomt, kan de uitbreiding verdwijnen. Omdat de $\succ\!\!<$-knoop grenzen en waaiers voor zich uit duwt, hebben we alleen regels nodig voor de interactie van een grens en een waaier met een variabele die een uitgebreid label heeft.

Op dezelfde manier als bij de rest van de regels geldt ook hier dat sommige regels alleen een schema weergeven: de waarde van een De Bruijn index is nul of groter, net als het uitgebreide label van een $\succ\!\!<$-knoop.

Met het splitsen van de uiteinden van de takken verdwijnt het gemeenschappelijke dat we vanwege de optimale stappen nodig hebben. Na t-interactie hebben we weer eenzelfde soort graaf als die welke e-interactie veranderde in een graaf waarin de interactie optimale stappen zet.

Een misverstand over betekenis

Met het beschrijven van een algoritme geven we een betekenis aan de objecten die het algoritme verandert. Zo geven we, in het geval van interactie in een graaf, aan waar de knopen in de graaf voor staan, en beschrijven we welk effect de knopen hebben wanneer we de graaf interpreteren.

De beschrijving van het eot-algoritme is bijzonder omdat we, wanneer we een betekenis aan knopen toekennen, alleen iets zeggen over andere knopen in de graaf, of over het effect dat knopen hebben zodra we een graaf als λ-expressie interpreteren. Dit is precies dat wat we oorspronkelijk met de beschrijving voor ogen hadden.

Omdat Lévy er waarde aan hecht dat het algoritme dat optimale stappen zet een plaats in de praktijk krijgt waarin we met λ-expressies werken, hebben we laten zien dat de beschrijving een dergelijke plaats rechtvaardigt. Niet in laatste plaats vanwege het feit dat correctheid het algoritme nodige en voldoende betekenis geeft.

We zien dus dat het helemaal niet nodig is om te proberen knopen een betekenis te geven door ze rechtstreeks met andere objecten in de praktijk te verbinden. Het zou dus een vergissing zijn als we los van correctheid een niet-duurzame betekenis aan knopen in grafen zouden toekennen.

Bibliografie

[1] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge University Press, 1998. 22, 27, 32, 83, 200, 202, 204, 207

[2] Andrea Asperti, Cecilia Giovannetti, Andrea Naletto. The Bologna Optimal Higher-order Machine. Technical Report UBLCS-95-9, Department of Computer Science, University of Bologna, March 1995. 203

[3] Richard S. Bird and Ross Patterson. De bruijn notation as a nested datatype. *Journal of Functional Programming*, 9:77–91, 1999. 116

[4] N.G. De Bruijn. Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. 16, 45

[5] Bert Jacobs en Jan Rutten. A tutorial on (co)algebras en (co)induction. http://www.cs.ru.nl/ bart/PAPERS/JR.pdf. 174, 176

[6] Georges Gonthier, Martín Abadi, Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '92, pages 15–26, Albuquerque, New Mexico, September 1992. ACM. 21

[7] Hai Liu and Paul Hudak. Plugging a Space Leak with an Arrow. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 193:29–45, november 2007. 205

[8] Dimitri Hendriks and Vincent van Oostrom. λ. In *Automated Deduction – CADE-19*, volume 2741 of *Lecture Notes in Computer Science*, pages 136–150. Springer Berlin / Heidelberg, 2003. 32, 117, 127

[9] J.-J. Lévy. Réductions Correctes et Optimales dans le Lambda-Calcul. Thèse de doctorat d'État, Universtité Paris VII. 1978. 11, 15

[10] John Lamping. An Algorithm for Optimal Lambda Calculus Evaluation. Technical Report Report Series SSL-89-27, Xerox PARC, Palo Alto, May 1989. 21

[11] John Lamping. An Algorithm for Optimal Lambda Calculus Reduction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 16–30, New York, NY, USA, 1990. ACM. 17, 32

[12] Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 95–108, New York, NY, USA, 1990. ACM. 35

[13] Ian Mackie. Efficient λ-Evaluation with Interaction Nets. In *Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 155–169. Springer Berlin / Heidelberg, 2004. 205

[14] Rudolf Carnap. *The logical syntax of language*. London: Kegan Paul [etc.], 1937. 173

[15] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, New York, 1971. 173

[16] Vincent van Oostrom, Kees Jan van de Looij, and Marijn Zwitserlood. *Lambdascope, Another optimal implementation of the lambda-calculus.* Workshop on Algebra and Logic on Programming Systems (ALPS), Kyoto, April 10th 2004. 28, 32, 203, 205

[17] Wadsworth, C.P. *Semantics and Pragmatics of the λ-calculus.* PhD. thesis, Oxford University, 1971. 16