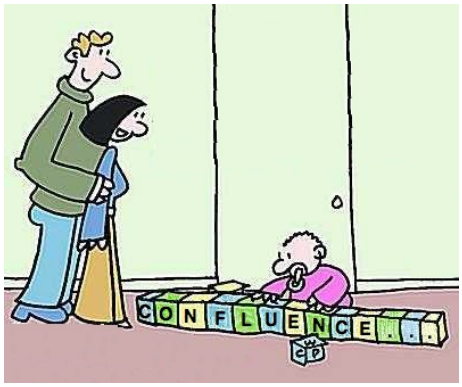
Confluence Constructor



Marieke Peeters
3151212

Master's Thesis
Department of Philosophy
University of Utrecht
45 ECTS

Thesis supervisors:
Dr. V. van Oostrom
Prof. Dr. A. Visser

Thesis reviewers:
Dr. V. van Oostrom
Prof. Dr. A. Visser
Prof. Dr. J.J.A. Meyer

10 februari 2010

Samenvatting

Dit artikel is geschreven als uitleg en gebruikershandleiding bij het softwareprogramma *Confluence Constructor*, dat een implementatie is van het constructieve bewijs van modulariteit van confluentie voor termherschrijfsystemen, zoals beschreven in *Modularity of Confluence Constructed* (2008) van Vincent van Oostrom ([5]). Het softwareprogramma is ontwikkeld door Marieke Peeters als afstudeeropdracht voor haar master Cognitive Artificial Intelligence aan de Universiteit Utrecht en is geschreven in Java. In dit artikel wordt eerst kort ingegaan op de achterliggende theorie van termherschrijfsystemen, waarna het geïmplementeerde bewijs zal worden toegelicht. Vervolgens zal de implementatie beschreven worden, en zal tevens een aantal keuzes, die tijdens de implementatie gemaakt zijn, nader worden toegelicht. Ook zal er een voorbeeld besproken worden, om de werking van het algoritme te verduidelijken. Ten slotte wordt er een gebruikershandleiding geboden bij het genoemde programma.

This article contains a manual that describes how the software-program Confluence Constructor, written in Java, can be used to construct common reducts for (modular) confluent trs's following the constructed proof of Vincent van Oostrom ([5]).

Inhoudsopgave

1 Voorwoord

Aangezien dit artikel niet alleen een gebruikershandleiding is, maar tevens een stageverslag dat toegankelijk is voor aankomende master-studenten, zal ik in dit voorwoord kort vertellen hoe ik tot dit project gekomen ben en hoe ik het ervaren heb.

Mijn afstudeertraject heeft een aantal wijzigingen gekend voor ik tot het schrijven van dit programma kwam. Ik wil Vincent van Oostrom en Noor Blaauw heel erg bedanken voor hun steun bij het vinden van het juiste pad voor mij. Zonder hen hadden de afgelopen jaren er heel anders uitgezien.

Ik ben via een omweg bij deze studie terecht gekomen. In mei 2006 studeerde ik af in de psychologie in Maastricht en kwam er achter dat ik nog iets anders wilde gaan doen: een studie met meer wiskunde en logica, een uitdaging. Zo kwam ik na wat zoekwerk op internet in contact met Noor. Met haar hulp ben ik begonnen aan een soort schakeljaar van 6 maanden, alvorens aan de master CAI te beginnen. In februari 2007 startte ik met mijn master. In eerste instantie volgde ik het traject Cognitive Dynamics, omdat dit dicht bij mijn vooropleiding lag, maar ik merkte gaandeweg dat ik de logica- en programmeervakken veel leuker vond. Daardoor ging ik steeds meer vakken in die richting volgen.

1.1 Afstuderen

Het moment kwam, waarop ik moest kiezen wat ik wilde gaan doen met het laatste jaar van mijn master CAI. Ik had in 2007 een bijeenkomst van Café KI bijgewoond, waar Vincent vertelde over zijn werk. Ik vond het erg interessant wat hij toen vertelde, en met die gedachte ging ik in april 2008 bij Vincent langs. Vincent vertelde me over zijn interesse in *term-herschrijfsystemen*. Vincent en ik bespraken verschillende onderwerpen die geschikt waren als afstudeerproject en kwamen uiteindelijk tot het volgende vraagstuk waar ik me een jaar mee bezig zou gaan houden:

Tijdens het schrijven van zijn artikel over Modulariteit van Confluentie [5] stuitte Vincent op een voorbeeld van twee herschrijfsystemen die niet aan Modulariteit van Confluentie voldoen. Ik zou gaan onderzoeken wat daar precies de oorzaak van is.

Nadat ik me twee maanden verdiept had in termherschrijfsystemen, besloot ik samen met Vincent, dat het onderzoeken van dit probleem als afstudeeronderwerp niet helemaal datgene was wat ik wilde doen. Samen bedachten we dat ik beter iets kon doen dat meer toepassingsgericht was, dan het theoretisch onderzoek waar we in eerste instantie voor gekozen hadden. We bedachten dat ik een Java-programma kon gaan schrijven dat Vincent's constructieve bewijs voor modulariteit van confluentie implementeert.

1.2 Verantwoording voor het huidige project en haar vorm

Het kennisgebied *termherschrijven* valt binnen de theoretische informatica, een domein dat van belang is voor de kunstmatige intelligentie, vanwege de kennis en het begrip van de programmatuurkunde dat oplevert. Door het vergroten van kennis op het gebied van theoretische informatica, wordt inzicht verkregen in de eigenschappen van programma's en de semantiek van functionele talen en constructies. Daarnaast verschaft de kennis van termherschrijven ook een hogere mate van inzicht in de equationele logica, wat over het algemeen gezien wordt als een belangrijk onderliggend vakgebied voor de kunstmatige intelligentie.

Aangezien mijn uiteindelijke afstudeerproject neerkwam op het ontwikkelen van een Java-implementatie van een constructief bewijs, is het resultaat van mijn afstudeerproject het beoogde programma. Hierdoor is het eindproduct van mijn master CAI geen gebruikelijke thesis, maar een softwareprogramma, voorzien van een inleiding en een gebruikershandleiding. Omdat deze laatste twee delen een tekstdocument vormen, worden deze twee tezamen hier voor het gemak toch benoemd als 'thesis', maar dit document kan niet los van het programma dat het beschrijft worden bekeken noch beoordeeld als resultaat van mijn afstudeerproject. In de inleiding staat de toedracht van het programma beschreven en de wijze waarop ik het behandelde probleem heb aangepakt. In de gebruikershandleiding staat beschreven hoe het programma gebruikt dient te worden. Het eindproduct bestaat dus uit drie delen: programma, verantwoording en handleiding. Deze drie delen tezamen vormen in mijn ogen een eindresultaat dat gepast is voor de opleiding CAI, omdat het getuigt van een verkregen inzicht en een directe praktische toepassing daarvan, zoals dat ook te zien is bij afstudeerprojecten van CAI-ers, die zijn uitgevoerd binnen het domein van de psychonomie. Daarbij wordt op basis van theoretische veronderstellingen een experiment uitgevoerd, waarvan de verslaglegging het eindproduct is. Een andere ge-

lijkende vorm van afstudeerprojecten, is het type projecten waarvan het onderwerp een simulatiemodel betreft. Overigens dient vermeld te worden dat aangezien het programma geschikt is voor internationaal gebruik, de gebruikershandleiding in het Engels is geschreven.

1.3 Indeling

De inleiding, sectie 2, bevat uitleg met betrekking tot de theoretische achtergrond en verantwoording van het programma Confluence Constructor. De eerste subsectie (subsectie 2.1) betreft een uitleg van termen, termherschrijven en termherschrijfsystemen. Daarop volgt een introductie van het concept confluentie in subsectie 2.2, modulariteit van confluentie en het belang van deze concepten voor het huidige onderwerp en voor de informatica in het algemeen. Vervolgens wordt het constructieve bewijs van modulariteit van confluentie [5] besproken in subsectie 2.3. In subsectie 2.4 treft u een beschrijving aan van de de complicaties die zich voordeden tijdens het implementeren, waarvoor de oplossingen niet direct volgden uit het constructieve bewijs. Hier vindt u tevens een beschrijving van de verantwoording voor de gekozen oplossingen. De inleiding wordt afgerond met een uitgewerkt voorbeeld in subsectie 2.5. Sectie 3 bevat de gebruikershandleiding voor het programma.

2 Inleiding

In deze gebruikershandleiding zal ik uitleggen waarvoor en hoe het besproken programma gebruikt kan worden. Alvorens de details van het programma te bespreken, is het wellicht handig te beschrijven binnen welke context het programma gebruikt kan worden. Hiervoor is het nodig een korte uitleg te geven van een gedeelte van de theorie van *Termherschrijven*.

In de komende paragrafen zal ik eerst bespreken wat termen zijn en hoe je deze kunt herschrijven. Ook zal ik omschrijven wat een termherschrijfsysteem is. Vervolgens zal ik ingaan op een bepaald type termherschrijfsystemen, te weten; confluente termherschrijfsystemen. En ten slotte zal ik uitleggen wat modulariteit van confluentie voor termherschrijfsystemen inhoudt. Dit alles is van belang voor het begrip van de werking van het in dit artikel beschreven programma.

2.1 Termherschrijven

In de voorgaande paragrafen is het concept al eens genoemd: termherschrijven. Maar wat is dat nu precies? Voor degenen die dat nog niet weten, wordt hieronder een korte uitleg gegeven. Belangrijk om te benadrukken is dat deze uitleg alleen bedoeld is voor mensen die, zonder enige voorkennis van termherschrijfsystemen, willen begrijpen wat dit programma doet. Voor een goed begrip van termherschrijven volstaat deze uitleg echter niet. Voor uitgebreidere informatie over termherschrijfsystemen, is het raadzaam standaard naslagwerken te bekijken, zoals [1], [2] of [3].

2.1.1 Termen

Voor een helder begrip van termherschrijven, is het goed om te beginnen met wat een term is. Termen zijn combinaties van functies en variabelen, waarbij voldaan moet worden aan het aantal argumenten dat een functie vraagt. Dit noemt men ook wel *welgevormd*. Ga maar na: iets optellen kan niet voordat duidelijk is wáár het bij opgeteld moet worden. Om te kunnen optellen, zijn er twéé dingen nodig. De functie + vraagt dan ook twee argumenten, oftewel + heeft een ariteit van 2. Een term wordt opgebouwd uit:

- Een verzameling functiesymbolen met een ariteit. De ariteit geeft aan hoeveel argumenten de functie nodig heeft. Een functie met een ariteit

van 0 heet ook wel een constante. Voorbeelden van functies zijn:

  · $+$ met ariteit 2

  · $\vee$ met ariteit 2

  · $\neg$ met ariteit 1

  · constanten $\top$, $\bot$, 2 en 10, allen met ariteit 0

- Een verzameling variabelen, bijvoorbeeld $\{x, y, z\}$.

Let op: De verzameling functiesymbolen en de verzameling variabelen hebben geen overlap, om ambiguïteit te voorkomen.

Een term lijkt in bepaalde opzichten veel op 'middelbare school-wiskunde'. Vaak wordt daar gewerkt met formules die bestaan uit functies, constanten en variabelen. Neem bijvoorbeeld de formule $breedte \times lengte$. In deze formule zijn *breedte* en *lengte* variabelen. De waarden van deze variabelen zijn niet gedefiniëerd en daarom kunnen ze nog alle waarden aannemen. Het symbool $\times$ is niet variabel, maar een gedefinieerde functie. $\times$ heeft twee argumenten nodig. In dit geval zijn dat *breedte* en *lengte*. In dit voorbeeld van een formule zou $breedte \times lengte$ als een term kunnen worden gezien. Deze zou er dan uitzien als $\times(breedte, lengte)$.

Andere voorbeelden van termen zijn:

- $+(2, 10)$ Dit is een prefix notatie: Eerst wordt de functie $+$ geschreven en dan de twee bijbehorende argumenten tussen haakjes. Hier op afgaande, zou $+(2(), 10())$ waarschijnlijk in de lijn der verwachting liggen, maar als een functie een constante is en dus geen argumenten nodig heeft, worden de haakjes weg gelaten. In *Confluence Constructor* wordt enkel gebruik gemaakt van prefix notatie.

- $\vee(x, \top)$ In deze term is een van de argumenten de variabele x.

- $\neg(\bot)$

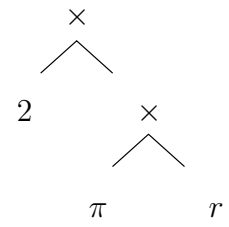
Voorbeeld 1: een term ontleden
Zie de formule $2 \times \pi \times r$. Hier zou $2 \times \pi \times r$ op twee manieren als term kunnen worden gezien : $\times(2, \times(\pi, r))$ of $\times(\times(2, \pi), r))$.
r (de straal van de cirkel) is in dit voorbeeld een variabele.
2 en π zijn hier constanten, deze waarden liggen al vast. Ze zijn gedefinieerd en kunnen geen andere waarde aannemen dan de waarde die ze volgens hun definitie hebben gekregen.
$\times$ is hier een functie op twee argumenten. De $\times$-functie komt twee keer

voor. De eerste $\times$ heeft als argumenten 2 en $\pi \times r$. De tweede $\times$ heeft als argumenten $2 \times \pi$ en r.

Een term kan worden opgesplitst in een zogenaamd *headsymbol* en *subtermen*. De headsymbol is de functie of variabele waar de term (in geval van prefixnotatie) mee begint. Subtermen zijn alle termen die aan te wijzen zijn in een term, inclusief de term zelf. In het voorbeeld $\times(2, \times(\pi, r))$, zijn de volgende subtermen aan te wijzen:

- de term zelf: $\times(2, \times(\pi, r))$
- de argumenten: 2 en $\times(\pi, r)$
- en de argumenten van de argumenten: π en r

Een term kan ook worden weergegeven als een boomdiagram. In Figuur 1 wordt de voorbeeldterm $\times(2, \times(\pi, r))$ weergegeven in een boomdiagram. Soms wordt een subterm uit een term gelicht. Het deel van de term waar



Figuur 1: Boomdiagram van de term $\times(2, \times(\pi, r))$

op dat moment niet naar gekeken wordt, heet ook wel de context. Stel dat we bij het voorbeeld $\times(2, \times(\pi, r))$ alleen geïnteresseerd zijn in de subterm $\times(\pi, r)$ en de rest van de term even willen laten voor wat het is. Dan noemen we $\times(2, \square)$ de context. Het symbool $\square$ heet *hole* en wordt gebruikt om in de context de positie aan te geven, waar de subterm, die we eruit lichten, stond.

Als laatste is het nog belangrijk op te merken, dat een substitutie een variabele vervangt door een term. Een substitutie geeft aan welke variabele vervangen wordt door welke term. Bijvoorbeeld: Neem de term $\times(x, x)$. Gegeven is de substitutie $\eta : x \mapsto 2$. Dan verandert de term $\times(x, x)$ onder substitutie η in $\times(2, 2)$. Elke variabele kan volgens een substitutie maar door één term worden vervangen, om ambiguïteit te voorkomen.

2.1.2 Termherschrijfsystemen

Een term kan worden herschreven volgens regels. Daarvoor hebben we termherschrijfsystemen nodig. In deze sectie zal ik uitleggen wat een termherschrijfsysteem is. Een termherschrijfsysteem wordt vaak afgekort tot trs, van *term rewriting system*. Een trs bevat functies en variabelen, zodat termen kunnen worden opgebouwd. Ook bevat een trs regels die vertellen hoe termen mogen worden veranderd oftewel herschreven. Soms worden voor een trs alleen de regels gegeven, met verschillende notaties voor functies en variabelen. Er mag dan vanuit worden gegaan, dat alleen de functies en variabelen die in de regels staan, onderdeel zijn van het systeem. Het herschrijven van een term wordt genoteerd met een pijltje. Een regel schrijft voor dat de term aan de linkerkant van de pijl veranderd mag worden in de term die aan de rechterkant van de pijl staat.

Voorbeeld 2: een termherschrijfsysteem
$+(x, 0) \rightarrow x$
Binnen deze trs zijn twee functies bekend: $+$, met ariteit 2, en 0, met ariteit 0 (een constante). Er is 1 variabele bekend binnen de trs, namelijk x. De regel zegt dat de term $+(x, 0)$ herschreven mag worden naar x. Dit lijkt logisch, want $x + 0$ is gelijk aan x.

Zie nogmaals Voorbeeld 1: $2 \times \pi \times r$. Zoals bekend, kan dit geschreven worden als twee verschillende termen: $\times(2, \times(\pi, r))$ of $\times(\times(2, \pi), r))$. Ook al is voor de meeste mensen duidelijk dat er hetzelfde uitkomt, is dat voor een systeem niet duidelijk. Er is een regel nodig die de twee termen gelijk maakt en een termherschrijfsysteem bevat zulke regels: Een trs geeft regels die voorschrijven hoe de ene term herschreven mag worden naar een andere term. Als $\times(2, \times(\pi, r))$ en $\times(\times(2, \pi), r))$ volgens een of meer regels kunnen worden herschreven naar een gelijke term, dan hebben ze een gemeenschappelijk reduct en zijn ze op een bepaald moment gelijk aan elkaar. In dit geval zou de volgende regel laten zien dat de termen gelijk zijn:
$\times(x, \times(y, z)) \rightarrow \times(\times(x, y), z)$

Als de linkerterm van een regel een variabele bevat, dan is die regel toepasbaar op alle termen die de vorm van die linkerterm hebben, met op de plaats van de variabele een willekeurige term. Ook mag de term die in de regel staat, voorkomen in een grotere term (een context). Dus, ervan uitgaande dat alle gehele getallen deel uitmaken van de trs uit Voorbeeld 2, kan de regel $+(x, 0) \rightarrow x$ worden toegepast op $+(2, 0)$, maar ook op $+(4, 0)$, en ook op $+(2, +(3, 0))$. De resultaten zijn dan:

$+(2,0) \to 2$ ($x \mapsto 2$)
$+(4,0) \to 4$ ($x \mapsto 4$)
$+(2,+(3,0)) \to +(2,3)$ ($x \mapsto 3$, en de context is $+(2,\Box)$)

Het toepassen van een regel op een term, het herschrijven van een term, wordt ook wel een reductie genoemd. Een reductie kan meerdere reductiestappen bevatten, waarbij een term meerdere keren wordt herschreven via de regels van de trs. Wanneer meerdere stappen achter elkaar worden gezet op een term, noemt men dit ook wel een reductierij. Een term die niet meer herschreven kan worden, heet een normaalvorm. In het geval van een simpele trs is het vaak vrij snel te zien hoe een kleine term gereduceerd kan worden naar zijn normaalvorm, zoals in Voorbeeld 3. In dit voorbeeld is het vrij eenvoudig zelf af te leiden wat het juiste antwoord is, maar wanneer een trs meer regels bevat en een term groter wordt, is het vaak niet meer zo eenvoudig om in één oogopslag te zien hoe een term gereduceerd wordt naar zijn normaalvorm.

Voorbeeld 3: een term reduceren naar zijn normaalvorm
Voor de volgende trs, reduceer de volgende term naar zijn normaalvorm: $+(S(S(0)),S(0))$. (De S kan gezien worden als de S van successor, wat opvolger betekent in het Engels. $S(S(0))$ betekent dus de opvolger van de opvolger van 0, en kan als 2 gezien worden. In feite gaat het in deze opgave dan om de som $2+1$.)
trs som:
$+(x,0) \to x$ (In 'normale' wiskunde: $x+0 \to x$)
$+(x,S(y)) \to S(+(x,y))$ (In 'normale' wiskunde: $x+(1+y) \to 1+(x+y)$)
Het antwoord is $+(S(S(0)),S(0)) \to S(+(S(S(0)),0)) \to S(S(S(0)))$ (In 'normale wiskunde': $2+1 \to 1+(2+0) \to 3$)

In het bovenstaande voorbeeld is er geen regel meer toepasbaar op de term $S(S(S(0)))$. Deze term is dus een normaalvorm. Er bestaan echter ook trs'en waarin er geen normaalvormen bestaan, zie Voorbeeld 4.

Voorbeeld 4: een oneindige reductie
Zie een trs met de volgende regels:
$\top \to \neg\bot$
$\neg\bot \to \top$
Een mogelijke reductierij in deze trs is:
$\top \to \neg\bot \to \top \to \neg\bot \to \top \to \neg\bot \to \top$

Bovenstaande reductie is oneindig. De term $\top$ kan met deze stappen altijd weer herschreven worden naar $\neg\bot$ en de term $\neg\bot$ kan altijd worden her-

schreven naar $\top$. Stel dat er aan bovenstaande trs de regel $\top \to \mathit{true}$ wordt toegevoegd. Dan zouden zowel $\top$ als $\neg\bot$ de normaalvorm *true* hebben, want $\top \to \mathit{true}$ en $\neg\bot \to \top \to \mathit{true}$ en *true* kan niet herschreven worden, aangezien er geen enkele regel is in de trs met aan de linkerzijde de term *true*.

Wanneer we nu ook de regel $\top \to \mathit{notfalse}$ toevoegen hebben $\top$ en $\neg\bot$ ineens twee normaalvormen. Ooit waren $\top$ en $\neg\bot$ gelijk (ze konden naar elkaar herschreven worden), maar het is mogelijk om twee verschillende regels toe te passen bij het reduceren van deze term. Afhankelijk van de keuze voor de regel $\top \to \mathit{true}$ dan wel de regel $\top \to \mathit{notfalse}$ in de reductie, is de normaalvorm *true* of *notfalse*. Deze twee termen kunnen niet meer naar elkaar herschreven worden en zijn dus niet meer gelijk, terwijl ze dat eerder in de reductie nog wel waren. In sommige trs'en is dit nooit mogelijk: wanneer je twee verschillende stappen toepast op een term die ooit gelijk was, dan kun je deze term altijd opnieuw gelijk maken. Dit brengt ons bij de volgende sectie.

2.2 Confluentie

Confluentie wil zeggen dat, wanneer een term door middel van verschillende stappen naar twee verschillende termen gereduceerd wordt (twee divergerende reducties), het mogelijk is die twee termen door het toepassen van andere stappen weer gelijk te maken.

Maar waarom is het nu eigenlijk interessant om confluentiebewijzen voor trs'en op te stellen? Om dit duidelijk te maken zal ik hier een kort uitstapje maken naar het 'grotere plaatje'. Om te beginnen is het binnen de wiskunde en de logica vaak niet zo interessant om voorbeelden van sommen uit te rekenen zoals in Voorbeeld 3. Meestal zijn wiskundigen meer geïnteresseerd in algemene uitspraken voor een willekeurig probleem met bepaalde eigenschappen. Dergelijke uitspraken zijn veel krachtiger, omdat ze een oplossing bieden voor een hele groep sommen, niet slechts voor één enkele som.

Stel dat Pythagoras had ontdekt dat er bij één driehoek toevallig geldt dat $a^2 + b^2 = c^2$. Dat was waarschijnlijk niet zo'n enorme doorbraak geweest in de wiskunde. Het interessante van de stelling van Pythagoras is juist dat deze voor èlke *rechthoekige* driehoek geldt. De stelling geeft een wetmatigheid voor driehoeken met bepaalde eigenschappen, namelijk voor rechthoekige driehoeken. Nu biedt de stelling een oplossing voor een

hele groep problemen, namelijk alle problemen die betrekking hebben op rechthoekige driehoeken.

Het is op een vergelijkbare manier interessant om wetmatigheden te ontdekken voor trs'en met bepaalde eigenschappen, zodat er al een aantal oplossingen geboden kan worden voor problemen die betrekking hebben op een bepaalde groep termherschrijfsystemen met bepaalde eigenschappen.

Een eigenschap van een trs geeft ten dele aan om wat voor soort trs het gaat, net als bij de driehoeken. Als bekend is dat het om een *rechthoekige* driehoek gaat, kan dat helpen bij het analyseren van het probleem en het vinden van oplossingen daarvoor. Vaak helpen eigenschappen bij het redeneren over problemen: ze geven aanknopingspunten voor het bedenken van een oplossing. Er zijn een hoop eigenschappen die trs'en kunnen hebben, maar in dit programma gaat het met name om de eigenschap van confluentie. Ik zal hieronder uitleggen wat confluentie in de praktijk kan betekenen.

Stel dat er een computerprogramma bestaat met een bepaalde procedure. Zie deze procedure als een soort stappenplan dat steeds vertelt welke keuzes er zijn, gegeven een situatie, en waar een keuze toe leidt. Het liefste wil de programmeur van dit stappenplan dat elke keuze en elke uitkomst duidelijk gedefinieerd is in het stappenplan. Het is niet wenselijk dat het stappenplan ergens in een eindeloze *loop* terecht komt, doodloopt of ambigu is. Het blijkt in de praktijk vaak lastig om dit te garanderen. Denk bijvoorbeeld aan het befaamde 'blauwe scherm', of een programma dat blijft 'hangen'. Dit was zeker niet de bedoeling van de programmeur, maar is een foutje dat over het hoofd werd gezien tijdens het schrijven van het stappenplan.

Confluentie van termherschrijfsystemen biedt echter een manier om te garanderen dat een procedure altijd hoogstens één uitkomst oplevert. Wanneer een procedure van een programma wordt omgezet naar een termherschrijfsysteem, en dat termherschrijfsysteem is confluent, dan betekent dat dat het termherschrijfsysteem altijd hoogstens één uitkomst levert voor een gesteld probleem en nergens ambigu is en geen doodlopende stappen bevat. En die eigenschap vertaalt zich terug naar de procedure, waarvan het termherschrijfsysteem was afgeleid.

Confluentie betekent dus dat wanneer een term naar twee verschillende termen gereduceerd wordt, deze twee termen altijd weer gelijk kunnen worden gemaakt door middel van 'gelijkmakende stappen', die ook wel convergerende reducties worden genoemd. Het resultaat hiervan heet het gemeenschappelijk reduct.

Voorbeeld 5: een voorbeeld van confluentie
Zie de volgende confluente trs.

trs 'gelijkheid':
$= (x, x) \to \top$ (gelijkheid controleren)
$\top \to \neg\bot$ (voor 'true' mag je 'niet false' schrijven)

We bekijken de volgende divergerende reducties:

$$\begin{array}{ccc} & = (\top, \top) & \\ & \swarrow \quad \searrow & \\ = (\neg\bot, \top) & & \top \end{array}$$

Omdat de trs confluent is, is het mogelijk om de twee verkregen reducten weer gelijk te maken via convergerende reducties. De convergerende reducties zijn:

$$\begin{array}{ccccc} = (\top, \top) & & \longrightarrow & & \top \\ \downarrow & & & & \downarrow \\ = (\neg\bot, \top) & \longrightarrow & = (\neg\bot, \neg\bot) & \longrightarrow & \top \end{array}$$

Figuur 2: voorbeeld 5.

Het is onbeslisbaar om voor elke willekeurige trs te bepalen of deze confluent is. Er zijn verschillende manieren om er achter te komen dat een trs confluent is. Geïnteresseerden wordt opnieuw aangeraden een van de eerder genoemde boeken te raadplegen([1],[2],[3]). Omdat het zo complex is om aan te tonen dat een trs confluent is, en omdat de complexiteit van dat probleem toeneemt wanneer de trs meer regels en symbolen bezit, kan het helpen wanneer de trs kan worden opgedeeld in kleinere delen. Op die manier is het mogelijk voor de kleinere delen te bepalen of ze confluent zijn. Maar voor je die methode kan gebruiken, moet er wel nog worden bewezen, dat de verzameling van deze kleinere delen nog altijd confluent is na samenvoeging. Dit probleem heet *Modulariteit van confluentie*.

2.2.1 Modulariteit van confluentie

Toyama gaf in 1987 ([6]) het eerste bewijs voor modulariteit van confluentie. Dit betekent dat:

Als twee (niet-overlappende) confluente systemen worden samengevoegd, dan is het resultaat van die twee systemen ook confluent. Ook geldt het omgekeerde: Als een confluent systeem in tweeën verdeeld wordt, zodanig dat er geen overlap bestaat tussen de symbolen van beide delen, dan zijn de afzonderlijke delen nog altijd confluent. (Met 'niet-overlappend' wordt bedoeld, dat er geen symbolen zijn die in beide systemen voorkomen.)

Voorbeeld 6: modulaire confluentie
Zie de volgende twee confluente trs'en.

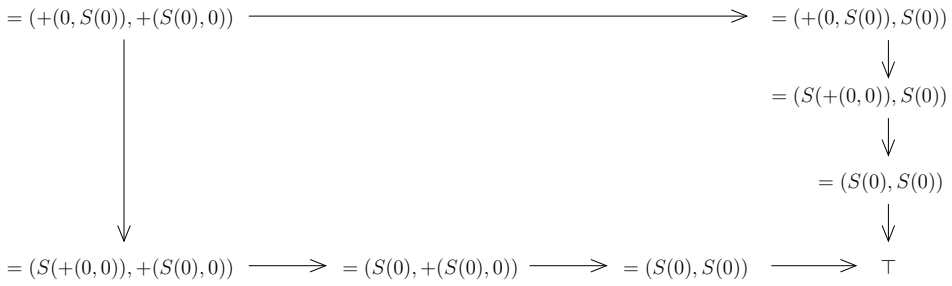
trs *som*:
$+(x, 0) \to x$
$+(x, S(y)) \to S(+(x, y))$

trs *if-equal-else* (*ite* staat voor if then else):
$ite(\top, x, y) \to x$
$ite(\bot, x, y) \to y$
$=(x, x) \to \top$

De gekozen divergerende reducties zijn:

$$\begin{array}{ccc} & =(+(0,S(0)),+(S(0),0)) & \\ \swarrow & & \searrow \\ =(S(+(0,0)),+(S(0),0)) & & =(+(0,S(0)),S(0)) \end{array}$$

Omdat de trs'en beide confluent zijn, is het mogelijk om de twee verkregen reducten weer gelijk te maken via convergerende reducties ([6]).
De convergerende reducties zijn:



Figuur 3: Voorbeeld 6: modulaire confluentie.

Vincent van Oostrom heeft in zijn artikel ([5]) een constructief bewijs geleverd voor deze stelling, dat naast een bewijs ook een basis voor een algoritme moest zijn, om het gemeenschappelijk reduct, de oplossing van het probleem, uit te rekenen. Een constructief bewijs is namelijk een bewijsvorm die het bestaan van een wiskundig object laat zien, door het te creëren of een methode te geven om zo'n object te kunnen creëren. Dit is een belangrijke tegenstelling met een non-constructief bewijs, dat het bestaan van een object bewijst, maar geen voorbeeld geeft van hoe dat object geconstrueerd kan worden. Veel non-constructieve bewijzen nemen aan het begin van het bewijs aan dat het object niet bestaat, en leiden vervolgens een tegenstelling af, met als conclusie dat het object wel moet bestaan. Ze maken dus gebruik van een absolute keuze: iets bestaat of het bestaat niet. Door te bewijzen dat het niet niet bestaat, wordt bewezen dat het wel bestaat.

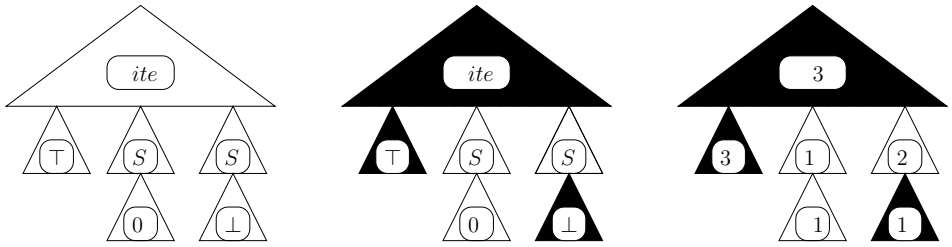
Confluence Constructor implementeert het bewijs uit het artikel ([5]). Het vindt dus, gegeven twee willekeurige confluente systemen met gemeenschappelijk reduct methoden, het gemeenschappelijk reduct berekenen voor twee divergerende reducties in het gecombineerde systeem en doet dat bovendien op een veel efficiëntere wijze dan door te zoeken met een *brute force algorithm*. Om terug te komen op de eerdere uitleg over de kracht van algemene uitspraken over willekeurige objecten met bepaalde bekende eigenschappen: Het constructieve bewijs voor modulariteit van confluentie laat zien *dat*, en bovendien *hoe*, voor elke willekeurige divergentie voor elke willekeurige verzameling confluente trs'en een gemeenschappelijk reduct geconstrueerd kan worden. Het geldt dus niet alleen toevallig voor de voorbeelden in deze gebruikershandleiding, maar voor elk willekeurig voorbeeld. Verwijzend naar de kracht van algemene uitspraken over objecten met bepaalde bekende eigenschappen, blijkt dit een krachtig en elegant bewijs voor modulariteit van confluentie.

2.3 Het constructieve bewijs

Voor de gedetailleerde technische versie van dit constructieve bewijs wordt verwezen naar het desbetreffende artikel, maar een en ander zal toch ook hier beschreven worden, om de implementatie te verduidelijken.

Een term uit een modulaire trs kan functies uit beide originele termherschrijfsystemen bevatten. In het volgende voorbeeld wordt opnieuw gebruik gemaakt van eerder gedefinieerde trs'en *som* en *if-equal-else*.

Voorbeeld 8: een rang geven aan een term
De term die wordt bekeken is $ite(\top, S(0), S(\bot))$. De term wordt als volgt opgedeeld in verschillende lagen van zwarte en witte delen: Subtermen afkomstig uit trs *som* zijn zwart gekleurd, en subtermen afkomstig uit trs *if-equal-else* zijn wit gekleurd. We delen de term op deze manier op in lagen van twee kleuren (zwart en wit). Variabelen krijgen altijd de kleur van de functie waar ze argument van zijn. Als alle delen gekleurd zijn, krijgen ze een rang. Subtermen bestaande uit één kleur krijgen rang 1. Subtermen bestaande uit twee lagen (één zwarte en één witte laag) krijgen rang 2. Subtermen krijgen dus als rang het maximaal aantal lagen van afwisselende kleuren. Voor een plaatje bij deze uitleg, zie Figuur 4.



Figuur 4: Modulariteit van confluentie: een term is afkomstig uit twee gecombineerde termherschrijfsystemen. De rang van de term is het maximaal aantal afwisselende lagen, in dit geval drie.

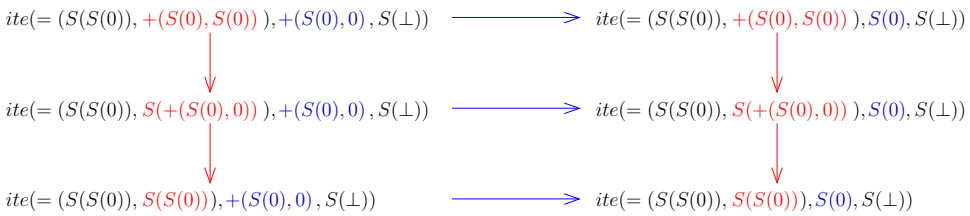
Interessante gegevens zijn hierbij dat subterm $S(0)$ in zijn geheel rang 1 krijgt, omdat deze subterm in zijn geheel afkomstig is uit trs som, terwijl subterm $S(\bot)$ rang 2 krijgt, omdat deze subterm uit meerdere lagen afkomstig uit verschillende trs'en bestaat. De gehele term krijgt rang 3, omdat de term uit maximaal drie alternerende lagen afkomstig uit de twee trs'en bestaat.

In het artikel wordt de term vervolgens aan de hand van de verkregen rangen

van de term en subtermen, opgesplitst in een zogenaamde *base* en *tall aliens*. De tall aliens zijn alle subtermen met de rang van de term minus 1. De base is de oorspronkelijke term met op de plek van de tall aliens een $\Box$. In het voorbeeld uit Figuur 4 is $S(\bot)$ met rang 2 een tall alien en blijft $ite(\top, S(0), \Box)$ over als base context.

Als duidelijk is welk deel of welke delen van de term tall aliens zijn en welk deel de base is, worden tall steps (stappen op de tall aliens) en short steps (stappen op de base) onderscheiden. In het voorbeeld is er geen enkele stap mogelijk op de tall alien, er zijn dus geen tall steps. Wel is er één stap mogelijk op de base, namelijk $ite(\top, S(0), S(\bot)) \to S(0)$. In dit voorbeeld is dat de enige mogelijke stap, en een short step.

De divergerende reducties in de modulaire trs worden geplaatst in een representatie van een reductiediagram. Een reductiediagram is een grafische weergave van de reducties op een term. In het geval van confluentie-bewijzen en -constructies wordt dit vaak getekend als een vierkante graaf die van linksboven naar rechtsonder loopt met éénzijdige pijlen. Linksboven staat de term die gereduceerd wordt. Langs de linkerzijde en de bovenzijde van het diagram lopen de divergerende reducties. Langs de onderzijde en de rechterzijde lopen de convergerende reducties. In het diagram zelf staan zogenaamde 'tegels' met divergerende en convergerende reducties voor elke tegel. Zie Figuur 5 voor een plaatje van een reductiediagram.



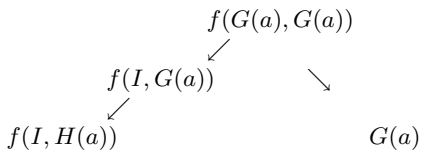
Figuur 5: Een reductiediagram: de linker- en bovenzijde van het diagram en de tegels zijn divergerende stappen. De rechter- en onderzijde bevatten de convergerende reducties.

In dit diagram is te zien hoe divergerende stappen worden gezet op de term $ite(= (S(S(0)), +(S(0), S(0))), +(S(0), 0), S(\bot))$. Zowel aan de linker- als de bovenkant van het diagram worden stappen gezet op de base. De stappen vinden in dit geval allemaal plaats binnen trs 'som'. Dit diagram bevat twee tegels, omdat de divergerende reductierij aan de linkerkant uit twee stappen bestaat en die aan de bovenkant uit één stap. Zo ontstaan er twee tegels die elk 'gesloten' worden door convergerende stappen aan de rechter- en onderkant van de tegel.

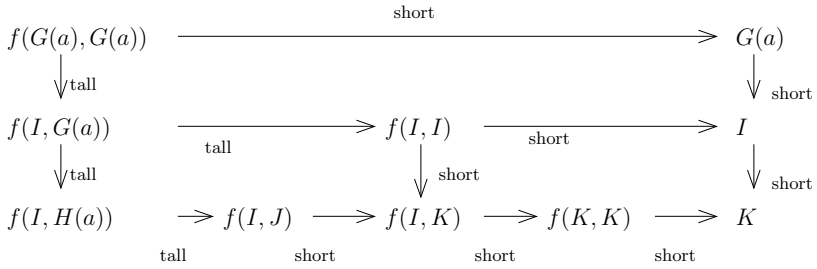
In de rest van deze handleiding zal een nieuw voorbeeld worden gebruikt, dat ook als *running example* voorkomt in het oorspronkelijke artikel([5]). Het is wellicht iets minder toegankelijk, omdat de functies abstract zijn, maar de abstractie van dit voorbeeld laat zien dat het constructieve bewijs van modulariteit van confluentie onafhankelijk is van de betekenis van de trs. De kracht van dit bewijs zit er juist in dat ongeacht de betekenis van de systemen en haar functies, het altijd mogelijk is dit bewijs op te stellen. De betekenis van de functies in de trs die hierna geïntroduceerd wordt, is ongespecificeerd (terwijl de S, de $+$ en *ite* (if then else) nog redelijk concreet waren). Deze samengestelde trs vormt een goed voorbeeld, om te laten zien welke moeilijkheden het probleem van modulariteit van confluentie zoal met zich mee brengt.

Voorbeeld 9: samengestelde trs voorbeeld-trs
Deze trs bestaat uit de volgende deel-trs'en:
t1 met functies a en f en regel $f(x,x) \to x$.
t2 met functies I, J, K, G en H en regels $G(x) \to I$, $I \to K$, $G(x) \to H(x)$, $H(x) \to J$ en $J \to K$.

Zie nu als probleem de divergerende reducties op term $f(G(a), G(a))$:



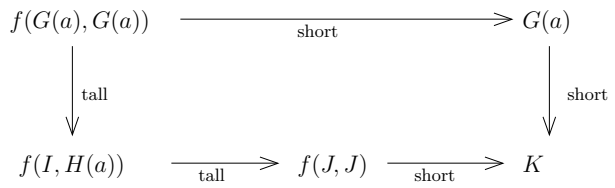
Zie voor een uitwerking van dit probleem Figuur 6.



Figuur 6: Een reductiediagram met zowel tall steps als short steps.

In het bovenstaande reductiediagram zijn alle stappen apart opgenomen, maar men zou ook in plaats van alle losse stappen te nemen, kunnen kijken naar reductierijen. Reductierijen zijn aaneengesloten stappen die achter

elkaar genomen zijn op een term. Omdat we te maken hebben met verschillende typen stappen is er echter één bijkomende restrictie: Een reductierij moet uit stappen bestaan van hetzelfde type: short of tall. Bekijk nogmaals eerder bekeken voorbeeld uit Figuur 6, maar ditmaal ingedeeld volgens haar reductierijtjes, zie Figuur 7.



Figuur 7: Een reductiediagram met zowel tall als short reductierijtjes. Dit diagram bevat één tall-short tegel.

Een extra type reductierij is de lege reductierij, waarin geen stappen gezet worden op een term. De noodzaak van dit type rij blijkt uit het geval waarbij de convergerende reductie aan de ene kant van de divergentie al klaar is en aan de andere kant nog niet. Hier komen we later nog op terug, bij het bespreken van belangrijke keuzes bij de implementatie. Voor nu is het voldoende om te stellen dat er een onderscheid kan worden gemaakt tussen zes verschillende tegels. Tegels worden geclassificeerd op de typen reductierijen die aan de linker- en bovenkant van een tegel staan (de divergerende reductierijen). De tegels hebben allemaal specifieke vormen van convergerende reductierijen. Deze zijn terug te vinden in Figuur 8 en worden verder uitgelegd als de Lemma's besproken zijn, verderop in dit artikel.

Er zijn twee categorieën tegels die in een reductiediagram voor kunnen komen, gebaseerd op de typen stappen die gezet kunnen worden. De eerste groep tegels bevat daadwerkelijke reducties, waarbij de termen aan beide zijden van de tegel veranderen.

1 Tegels waarbij de reductierijen aan beide zijden van de divergentie van type short zijn.

2 Tegels waarbij de reductierijen aan beide zijden van de divergentie van type tall zijn.

3 Tegels waarbij de reductierijen aan één zijde van de divergentie van type short zijn en aan de andere zijde van de divergentie van type tall zijn.

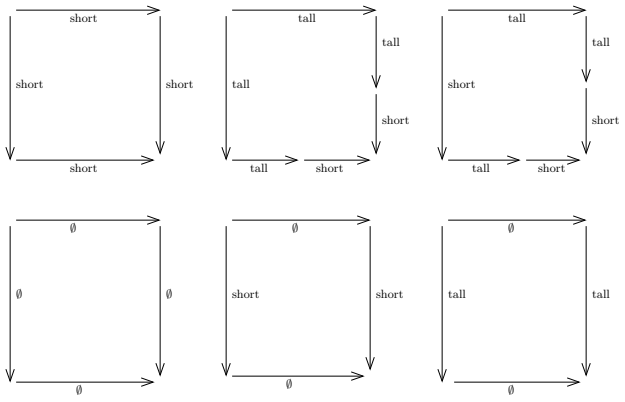
Dan zijn er nog tegels waarbij aan één of twee zijden van de tegel geen

echte stappen gezet worden, maar zogenaamde lege stappen: de term verandert aan minstens één zijde van de tegel niet. Dit noemt men ook wel gedegenereerde tegels.

4 Tegels waarbij de reductierijen aan beide zijden van de divergentie leeg zijn.

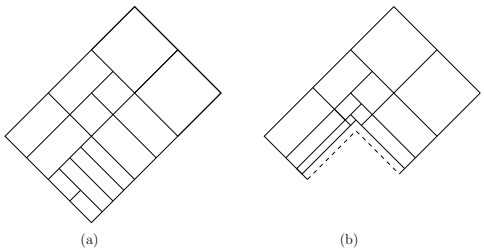
5 Tegels waarbij de reductierijen aan één zijde van de divergentie van type short zijn en aan de andere zijde van de divergentie leeg zijn.

6 Tegels waarbij de reductierijen aan één zijde van de divergentie van type tall zijn en aan de andere zijde van de divergentie leeg zijn.



Figuur 8: De 6 verschillende typen tegels en hun convergerende eigenschappen.

Belangrijk is het hierbij om te benadrukken, dat het constructie-proces uiteindelijk zal stoppen. Dit wil zeggen, dat er uiteindelijk een einde komt aan het tegelen van de convergerende reducties, zie ook Figuur 9. Voor meer informatie over deze uitspraak en het bewijs ervan met behulp van Decreasing Diagrams, wordt verwezen naar twee artikelen ([4], [5]).



Figuur 9: Voorbeeld van een diagram waarbij het tegelen eindigt (a) en waarbij het tegelen oneindig doorgaat (b).

Bij het sluiten van de verschillende typen tegels wordt gebruik gemaakt van drie lemma's. Hieronder zullen de drie lemma's geïntroduceerd worden. De voorbeelden die erbij gegeven worden komen uit *voorbeeld-trs*. In Figuur 6 zijn de stappen uit de eerste twee voorbeelden die hier genoemd worden terug te vinden.

Lemma 1 De base-context kan gereduceerd worden met een substitutie voor de tall aliens.

Lemma 2 Als twee tall aliens gelijk waren, dan moeten ze gelijk blijven, dus pas dezelfde stappen toe op gelijke tall aliens. Dit wordt opgelost door bij te houden welke tall aliens eerder in het reductiediagram gelijk waren en vervolgens ófwel, wanneer slechts één van de tall aliens gereduceerd is, dezelfde stappen die op de ene tall alien zijn toegepast, ook toe te passen op de andere tall alien, ófwel, wanneer beide tall aliens naar verschillende termen gereduceerd zijn, een gemeenschappelijk reduct uit te rekenen voor de twee tall aliens.

Lemma 3 Tall aliens kunnen gereduceerd worden binnen de base-context.

Voorbeelden van deze drie lemma's:

1 $f(G(a), G(a)) \to G(a)$. De base-context $f(\Box, \Box)$ wordt hier gereduceerd naar $\Box$ door toepassing van de regel $f(x, x) \to x$ uit trs 1 van voorbeeld-trs. De tall aliens worden in eerste instantie uit de term gehaald, de convergerende stappen op de base worden geconstrueerd en in het resultaat daarvan worden de tall aliens terug geplaatst met behulp van substitutie $x \mapsto G(a)$.

2 Voorbeeld voor het eerste geval: $f(G(a), G(a)) \to f(I, G(a))$. De twee gelijke tall aliens $G(a)$ worden ongelijk gemaakt door toepassing van de regel $G(x) \to I$. Om beide tall aliens weer gelijk te maken wordt de stap $G(a) \to I$ gekopieerd naar de tweede tall alien $G(a)$ met als resultaat de stap $f(I, G(a)) \to f(I, I)$. Voorbeeld voor het tweede geval: $f(G(a), G(a)) \to f(I, G(a)) \to f(I, H(a))$. Hier zijn beide tall aliens gereduceerd naar verschillende tall aliens. In dit geval zal er recursie worden toegepast op de tall aliens om het gemeenschappelijk reduct uit te rekenen voor I en $H(a)$. Uit deze recursie zal blijken dat zowel I als $H(x)$ naar J reduceren, waardoor het gemeenschappelijk reduct J zal zijn en de term naar $f(J, J)$ gereduceerd zal worden alvorens verder te gaan met de oorspronkelijke divergentie.

3 Eigenlijk was hierboven in het voorbeeld van Lemma 2 al te zien, dat de tall aliens uit de context werden genomen en na de convergentie terug werden geplaatst. Toch volgt hier voor alle helderheid nog een voorbeeld van Lemma 3: $f(J, H(a)) \leftarrow f(J, G(a)) \rightarrow f(J, I))$. In dit geval vindt een reductie plaats in de context $f(J, x)$ met substitutie $x \mapsto G(a)$. $G(a)$ wordt aan de ene kant van de divergentie gereduceerd naar $H(a)$ en aan de andere kant naar I. De context blijft onveranderd. Die kan dus tijdelijk worden gestript van de tall alien waar het om draait en vervolgens is het mogelijk om de divergentie op de tall alien op te lossen en later terug te plaatsen in de context. Zo ontstaat de convergentie $H(a) \rightarrow J \leftarrow I$. Die in de context geplaatst kan worden, met als resultaat $f(J, H(a)) \rightarrow f(J, J) \leftarrow f(J, I)$.

Deze drie lemma's geven samen de oplossing voor alle mogelijke tegels, die te zien zijn in Figuur 8. Terugkomend op de convergerende eigenschappen van de verschillende tegels, kunnen we nu stellen dat in het geval van een short–short tegel, het met Lemma 1 mogelijk is de tall aliens tijdelijk uit de base te halen, en de 'gestripte' base te convergeren, waarna de tall aliens terug worden geplaatst volgens de substitutie. In geval van een tall–tall tegel worden de tall aliens uit de base gehaald en worden juist deze tall aliens geconvergeerd, waarbij gelijke aliens gelijk worden gehouden met behulp van Lemma 2. Daarna worden de tall aliens teruggeplaatst in de base context met Lemma 3. Het ingewikkeldste geval, de short–tall tegel, kan worden opgelost met behulp van alle drie de Lemma's. Het komt er op neer, dat de stappen op de base en de stappen op de tall aliens over elkaar geprojecteerd worden, onder behoud van gelijke tall aliens. Wanneer een tegel lege rijtjes bevat, worden de stappen aan de niet-lege kant geprojecteerd over de lege rij. In het geval van de tegel met aan beide zijden lege rijtjes gebeurt er niets.

2.4 Implementeren

In de komende sectie zal dieper in worden gegaan op de technische aspecten van het implementeren van het bewijs. Hiervoor is meer kennis vereist van termherschrijfsystemen, dan dat in de inleiding geboden wordt. Er wordt aangeraden één van de eerder genoemde naslagwerken te raadplegen wanneer een en ander niet meteen helder is. Het hier beschreven softwareprogramma is geschreven in Java, een object-geöriënteerde programmeertaal die ons de mogelijkheid bood tot het creëren van objecten zoals termen, termherschrijfsystemen en confluentiediagrammen. Voor de geïnteresseerden die niet bekend zijn met Java biedt [7] een goede uitleg.

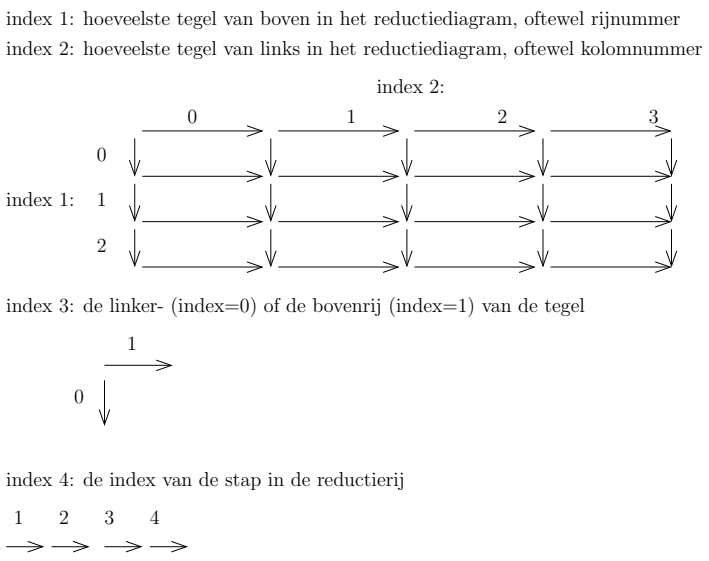
Confluentieconstructie voor niet-samengestelde trs'en

Het programma bevat methodes die verschillende gemeenschappelijk reduct methodes benutten voor verschillende typen confluente systemen. Om te beginnen kunnen trs'en samengesteld, modulair, zijn of niet-samengesteld. Samengestelde trs'en zijn zoals eerder verteld confluent wanneer de trs'en waaruit ze bestaan confluent zijn, maar hoe weten we of de deze niet-samengestelde trs'en confluent zijn? Niet-samengestelde trs'en kunnen om twee standaardredenen confluent zijn. De ene reden is dat de trs Sterk Normaliserend (SN) en Weak Church-Rosser (WCR) is. Maar ook wanneer een trs orthogonaal is, is de trs confluent. De gemeenschappelijk reduct methodes voor deze twee typen confluente systemen verschillen, en voor beide typen zijn dus afzonderlijke methodes geschreven in het programma. In het geval van een trs die voldoet aan de eigenschappen SN en WCR, worden de termen, die uit de divergentie resulteren, beide naar hun normaalvorm gereduceerd. Vervolgens wordt gekeken of deze normaalvormen gelijk zijn. In het geval van een orthogonale trs wordt gebruik gemaakt van de methode van de onderlijnde trs. Voor een duidelijke uitleg van deze methoden wordt opnieuw verwezen naar een van de eerder genoemde naslagwerken ([1], [2] of [3]).

Confluentieconstructie voor orthogonale trs'en

Bij het ontwikkelen van de methode voor confluentieconstructie voor orthogonale trs'en, deden zich twee interessante moeilijkheden voor, die we hieronder verder zullen toelichten. Ook zullen we de oplossingen die we

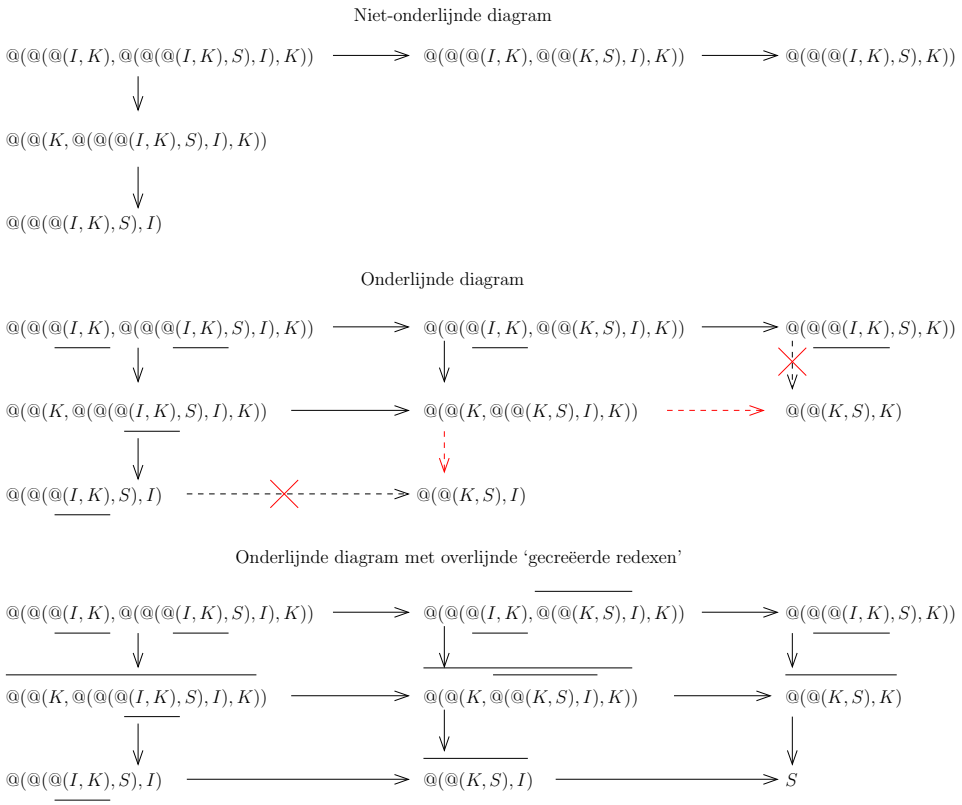
bedacht hebben bespreken. De eerste moeilijkheid die ontstond, was het bedenken van een handige representatie voor een reductiediagram. We hebben uiteindelijk gekozen voor een representatie met gebruikmaking van een vierdimensionale ArrayList. In Figuur 10 is weergegeven hoe de vier indexen de representatie geven van een reductiediagram. Deze representatie is ook gebruikt als representatie van het reductiediagram bij het construeren van het gemeenschappelijk reduct voor modulaire confluente trs'en, waar in de volgende sectie dieper op in zal worden gegaan.



Figuur 10: De betekenis van de vier indexen van de ArrayList-representatie van een reductiediagram bestaande uit tegels van reductierijtjes.

De tweede moeilijkheid die we tegenkwamen bij het schrijven van de methode voor confluentieconstructie van orthogonale systemen, was hoe we het oorspronkelijke reductiediagram konden optillen naar het onderlijnde reductiediagram. Daarbij was het belangrijk de redexen door te geven en bovendien moesten de in de divergentie nieuw gecreëerde redexen aan het diagram worden toegevoegd, om de uiteindelijke onderlijnde convergerende stappen te vinden. Wat we gedaan hebben om dit voor elkaar te krijgen, is het volgende: Het reductiediagram wordt opgesteld met de normale, niet-onderlijnde, reductiestappen. Vanuit dit reductiediagram wordt het onderlijnde reductiediagram opgebouwd, en wel op de volgende manier: De beginterm, de term die linksboven in het reductiediagram staat, wordt onderlijnd en ook de oorspronkelijke trs wordt onderlijnd, zoals beschreven staat in de

eerder genoemde standaard naslagwerken. Vervolgens worden alle mogelijke reductiestappen binnen de onderlijnde trs op deze onderlijnde beginterm uitgerekend. Hierna worden de twee divergerende stappen op deze term in het oorspronkelijke diagram vergeleken met de gevonden mogelijke onderlijnde reductiestappen voor de onderlijnde versie van deze term. De onderlijnde versies van de niet-onderlijnde stappen worden gevonden en in het onderlijnde diagram ingevuld. Dan wordt voor de tegel de convergerende reductie geconstrueerd.



Figuur 11: Een voorbeeld van een confluentie-probleem binnen de orthogonale trs CL, met gebruikmaking van recursie op de onderlijnde trs. Omdat de nieuwe redexen in het tweede diagram nog niet zijn toegevoegd is het niet mogelijk het gehele diagram in te vullen. Na toevoeging van de nieuwe redexen is dit wel mogelijk.

Vervolgens wordt recursief voor elke tegel bepaald of de stap, in het oorspronkelijke diagram, plaatsvindt op een onderlijnd deel van de term, een 'bekende' redex, in het onderlijnde diagram. Zo niet, dan wordt dat deel alsnog als redex onderlijnd. De rest van de procedure is weer als voor-

heen: alle mogelijke reducties op de resulterende onderlijnde term (met extra redex) worden uitgerekend en de oude onderlijnde stappen (zonder de nieuwe redex) worden vervangen door de equivalente stappen met de extra redex. Dan kunnen voor elke tegel weer de convergerende stappen worden uitgerekend en worden ingevuld in het diagram. Wanneer de buitenzijdes van het diagram bekend zijn, hebben we de uiteindelijke convergentie voor het hele diagram. Voor een voorbeeld van een confluentieprobleem in de trs van combinatorische logica (CL), zie Figuur 11.

In dit voorbeeld ontstaan aan beide zijden van de divergentie nieuwe redexen door latere stappen in de reductierijen. Deze nieuwe stappen worden in het diagram aan de bovenzijde van de term 'overlijnd'. In de figuur is ook te zien dat de nieuwe redexen gekopieerd worden in de rest van het reductiediagram, waardoor het mogelijk is de convergerende stappen te construeren.

De regels van CL:
$@(@(@(S,x),y),z) \to @(@(x,y),@(x,z))$
$@(I,x) \to x$
$@(@(K,x),y) \to x$

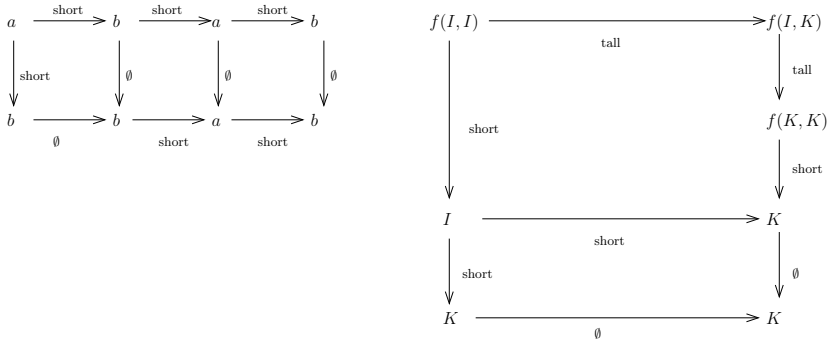
Als eerste wordt de term $@(@(@(I,K),@(@(@(I,K),S),I),K))$ onderlijnd en worden alle mogelijke stappen op deze term in de onderlijnde trs uitgerekend. De onderlijnde versies van de oorspronkelijke stappen worden gezocht en geconvergeerd. Zo wordt tegel (1,1) geconstrueerd. Vervolgens wordt doorgegaan met de volgende tegel, maar deze stap wordt genomen op een subterm die eerst nog geen redex was. Dus wordt deze subterm met terugwerkende kracht onderlijnd als redex. Nu is het mogelijk deze onderlijnde term te reduceren en de onderlijnde versies van de divergerende stappen te vinden. Deze worden weer ingevuld in het diagram en geconvergeerd. En zo wordt het hele diagram ingevuld, tot het gemeenschappelijk reduct gevonden is.

Confluentieconstructie voor samengestelde trs'en

Toen de confluentie-methodes voor de afzonderlijke typen confluente trs'en klaar waren, kon de stap gemaakt worden naar de modulaire termherschrijfsystemen. We volgden hierbij het eerdergenoemde artikel ([5]): Eerst wordt een term volgens de rang-berekening gesplitst in de base en de tall aliens. Vervolgens worden de stappen ingedeeld als tall en short steps en ingevuld in een reductiediagram-representatie zoals eerder uitgelegd in Figuur 10. Dan

wordt recursief per tegel gekeken welke methode (en welke van de drie eerder besproken lemma's) gebruikt moet worden om de convergerende stappen te berekenen voor die tegel met behulp van de classificatie van tegels, uitgelegd in Figuur 8. Dit alles gebeurt met een volledige inductie op de rang, waardoor uiteindelijk op rang 1 de afzonderlijke gemeenschappelijk reductmethoden ingezet kunnen worden, voor de orthogonale trs of voor de trs die voldoet aan de eigenschappen van WCR en SN.

Bij het schrijven van de methode voor confluentieconstructie van modulaire trs'en kwamen we nog twee moeilijkheden tegen. De eerste moeilijkheid waar we tegenaan liepen was de noodzaak van het definiëren van lege rijtjes. In het artikel vormt dit geen probleem, omdat er niet zo expliciet gewerkt wordt met symmetrische, vierkante reductiediagrammen, wat wel het geval is in het programma. Maar in de implementatie moest er een aparte definitie komen voor het lege rijtje om het reductiediagram af te kunnen maken, wanneer aan één kant van de tegel het gemeenschappelijk reduct al gevonden was. Om dit op te lossen, hebben we ervoor gekozen om naast het type *tall step* en *short step*, nóg een type stap, namelijk de *empty step*, aan te maken. *Empty steps* bevatten een 'regel' $x \to x$ en hebben als start- en eindterm dezelfde term. Voor twee voorbeelden van het gebruik van de lege reductiestap, zie Figuur 12. Lege reductiestappen worden aan het eind van de confluentieconstructie verwijderd uit de reductierijen en zijn dus niet terug te vinden in de resulterende output in de GUI.

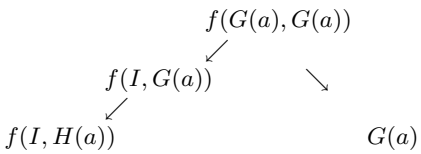


Figuur 12: Twee voorbeelden ter illustratie van de noodzaak voor het definiëren van lege reductiestappen.
a) binnen het non-modulaire orthogonale systeem $\{a \to b, b \to a\}$
b) binnen het running example, de modulaire trs *voorbeeldtrs*.

De andere moelijkheid was het vinden van een manier om de zogenaamde *balancing steps* op te slaan. Balancing steps zijn de stappen die ervoor moeten zorgen dat aliens die ooit gelijk waren, opnieuw geconvergeerd worden en dus weer gelijk worden gemaakt, zodat gelijkheid behouden blijft. De oplossing waarvoor we gekozen hebben, is als volgt: Voor elke alien worden vier arraylists van reductiestappen bijgehouden, twee voor de linker- en bovenkant van de divergerende stappen op de alien en twee voor de onder- en rechterkant van de convergerende stappen op de alien. Op deze manier wordt voor elke alien bijgehouden hoe deze gereduceerd wordt en zijn de afzonderlijke reductierijen per alien aanspreekbaar, wat de projectie van stappen vereenvoudigt. Deze methode is in feite een zeer expliciete uitvoering van Lemma 3.

2.5 Een uitgewerkt voorbeeld van een tall-short tegel

Bekijk nogmaals de divergerende reducties op term $f(G(a), G(a))$ binnen *Voorbeeld-trs* uit Voorbeeld 9:

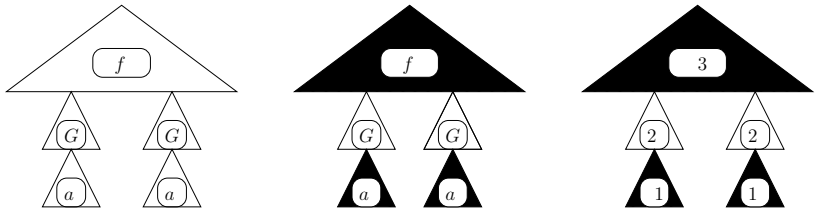


Dit voorbeeld is tevens het *running example* uit het artikel([5]).

De output van het programma zoals die in de GUI wordt gegeven, is niet de complete berekening die uitgevoerd wordt door het programma. Het programma geeft slechts de convergerende reducties aan de buitenzijde van het diagram, maar daarbinnen zijn een aantal andere reducties uitgerekend. Hieronder staat een beschrijving van de complete berekening van het voorbeeld, zoals uitgevoerd door het programma.

Als eerste wordt de rang van de term berekend. De term heeft rang 3, zie Figuur 13, en daardoor is bekend dat de term niet kan worden opgelost door middel van een van de gemeenschappelijk reduct methoden die voor niet-samengestelde trs'en bekend zijn. De term moet dus worden opgelost door de gemeenschappelijk reduct methode voor modulaire trs'en.

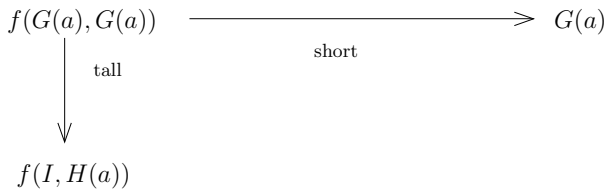
Vervolgens wordt gekeken wat de stappen zijn waaruit de divergentie bestaat en of dit tall steps of short steps zijn. Als één zijde van de divergentie uit zowel short als tall steps bestaat, wordt deze opgedeeld in verschillende



Figuur 13: De term $f(G(a), G(a))$ heeft rang 3.

tegels. In dit voorbeeld is er, vanuit de startterm, maar één tall-short tegel, zie Figuur 14.

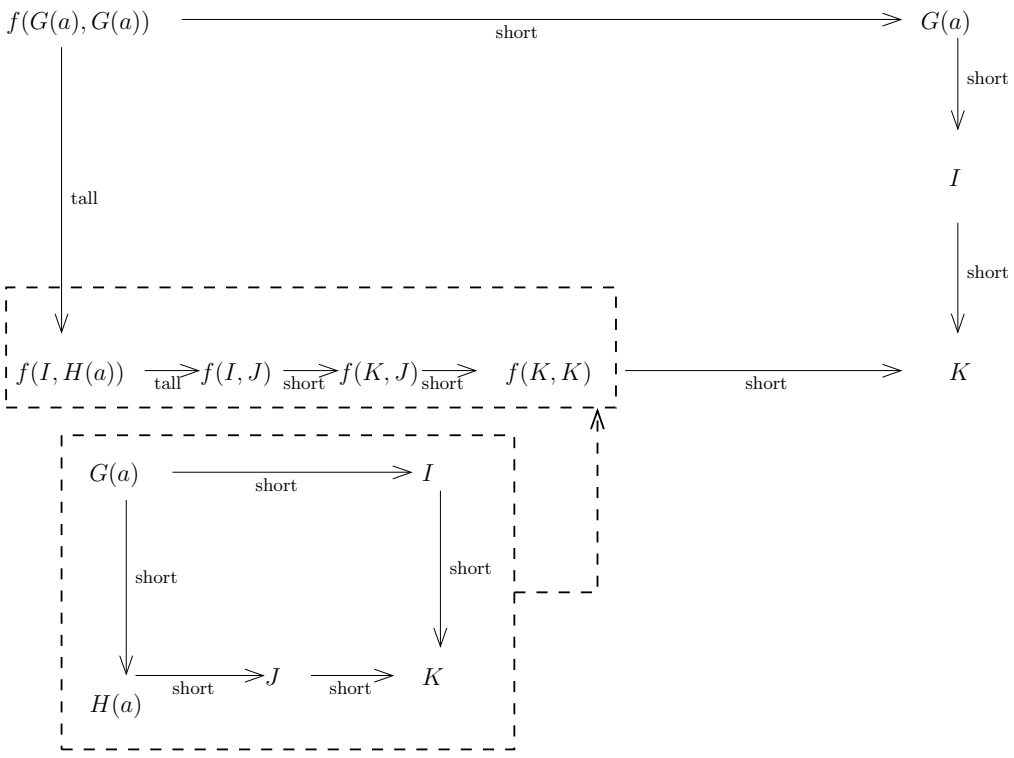
De tegel wordt als zodanig herkend en de bijbehorende methode wordt aangeroepen. Aan de kant van de tall steps, vindt balancing plaats, met behulp van Lemma 2. De twee tall aliens worden gelijk gemaakt door hun gemeenschappelijk reduct uit te rekenen. De twee stappen op de verschillende tall aliens worden uit hun context gehaald, met behulp van Lemma 3, Waardoor de divergentie op de tall aliens er als volgt uit komt te zien: $I \leftarrow G(a) \rightarrow H(a)$. Dit resulteert in het diagram dat linksonder te zien is in Figuur 15.



Figuur 14: De originele divergentie $f(I, H(a)) \leftarrow f(I, G(a)) \leftarrow f(G(a), G(a)) \rightarrow G(a)$ bestaat uit één tall-short tegel.

Deze divergentie wordt recursief naar de modulaire gemeenschappelijk reduct methode gestuurd. Daar wordt opnieuw de rang van de startterm ($G(a)$) bepaald, en deze is 2. Dit betekent dat er nog altijd geen gebruik kan worden gemaakt van de gemeenschappelijk reduct methoden voor niet-samengestelde trs'en. De stappen op $G(a)$ zijn beiden van type *short*, daarom wordt deze divergentie geconvergeerd met behulp van Lemma 1.

De stappen op de base worden bekeken als stappen op een context. De tall aliens worden uit de context genomen en gesubstitueerd door variabelen. Op die manier houdt men de volgende divergentie over: $I \leftarrow G(x_0) \rightarrow H(x_0)$ met substitutie $\eta : x_0 \mapsto a$. Omdat de startterm van deze divergentie,



Figuur 15: De eerste recursie bij het construeren van het gemeenschappelijk reduct voor voorbeeld $f(I, H(a)) \leftarrow f(I, G(a)) \leftarrow f(G(a), G(a)) \rightarrow G(a)$: Balancing van de tall aliens.

$G(x_0)$, van rang 1 is, kan deze divergentie worden opgelost door een gemeenschappelijk reduct methode voor niet-samengestelde trs'en, afhankelijk van de eigenschappen van de trs. In dit geval is $G(x_0)$ afkomstig uit *t2* en deze is WCR en SN. De divergentie wordt dus uitgerekend door de methode ConfluenceWCRSN met als resultaat de convergerende stappen $I \rightarrow K \leftarrow J \leftarrow H(x_0)$, zie ook Figuur 15. Deze stappen worden terug geplaatst in de base context, waardoor de volgende stappen in het 'grote' diagram komen te staan: $f(I, H(a)) \rightarrow f(I, J) \rightarrow f(K, J) \rightarrow f(K, K)$. Vervolgens wordt de *short* stap geprojecteerd naar de zijde van de *tall* stappen, met als resultaat: $f(K, K) \rightarrow K$. Dit is mogelijk, doordat balancing plaatsgevonden heeft!

Als laatste worden de alien stappen geprojecteerd op de base. Op de term $G(a)$ is binnen de context $f(\Box, G(a))$ de reductie $G(a) \rightarrow I \rightarrow K$ toegepast. Deze reductie wordt aan de zijde met de 'short' stap van de tegel geprojecteerd. De reden waarom deze reductie geselecteerd wordt en niet de reductie

$G(a) \to H(a) \to J \to K$ die binnen de context $f(G(a), \Box)$ is toegepast, is omdat deze reductie simpelweg is toegepast op een alien die later in de term voorkomt en het algoritme selecteert de eerst gevonden reductierij op de desbetreffende tall alien. Met de projectie van deze reductie naar de short-zijde van de tegel is de convergentie berekend en is het gemeenschappelijk reduct, K, gevonden.

3 User's Manual

The goal of this section is to inform the user about the less obvious aspects of the program. In general the program is straightforward in its use and the labels in the GUI are clear about the purpose of the buttons and textfields. Nevertheless there are a few restrictions with respect to the input in the textfields. Because of that we will give a short guide to the use of this program in this section. Explanations about input-restrictions will be marked with a ★. It was a conscious choice to keep the user's manual brief and to the point.

Double-click the executable jar-file 'Confluence Constructor.jar', to start the program. Subsequently click the button 'start program'.

3.1 Creating a new trs

To add a new trs to the program, click 'build/edit TRS's', followed by the 'Build new TRS'-button. After that it is possible to go back and forth between five screens representing the five features of the trs that are to be defined: its name, its variables, its functions, its rules and the assumptions that are valid in the trs. To add a new part of the trs, enter the required information in the textfield(s) and click the 'add'-button. To remove an incorrectly added part, check the checkbox in front of the incorrect part and click the 'remove checked and continue'-button.
★ When entering new terms to create rules, mind the next restrictions:

- all symbols used in the terms need to be added to the trs, including the variables!
- the terms need to be well-formed: they need to obey the arity of the functions by giving the correct number of arguments for each function.
- the terms are to be entered without parentheses and in prefix notation.

At the end of the sequence of screens, click the button 'Finish and build trs'. The new trs will be created and will appear in the upper left corner of the main panel and the main editing panel, where all trs's known to the program can be found.

3.2 Modifying an existing trs

To modify an existing trs from the main menu, click the button 'build/edit TRS's'. In the 'Build and/or edit' screen, check the box in front of the trs that is to be modified and click the 'edit'-button. In the next screen, check the features of the trs that are to be modified. By clicking 'edit', a series of screens will follow in which it is possible to choose the parts of the trs that are to be changed and in which way they are to be changed. In each screen it is possible to go back to the 'select features'-screen to adjust the selection of features that are to be modified, or to stop editing and go back to the main editing panel.
★ When entering terms to change the set of rules, mind the restrictions discussed above in the section 'creating a new trs'. Moreover, keep in mind that if the arity of functions is changed, the rules need to be changed accordingly!

3.3 View the features of an existing trs

To look at the features and parts of an existing trs in the main menu or the main editing menu, check the box in front of the trs in the top left corner and click the 'view'-button. A pop-up window will appear showing the details of the selected trs. It is only possible to view one trs at the same time.

3.4 Constructing a confluence proof

To compute the common reduct for a given divergence of a term coming from a (modular) trs, go to the main menu and check the boxes in front of the two trs's in which the reductions will take place. In the textfield in the top right corner, enter the term that is to be reduced. In the textfields below that, enter the number of reductionsteps that are to be taken on the left and rightside of the divergence. When done entering the required information, click 'Confluence'.
★ When entering the term, mind the restrictions as mentioned before in the

section ‘Creating a new trs’.
★ When deciding the amount of steps to be taken upon the term, do not exceed the maximum possible number of reductionsteps applied to the term within the chosen trs’s.
After clicking the ‘Confluence’-button, a sequence of screens will follow, in which the diverging steps can be defined. In each screen, select the exact reduction-step to put in the divergence. The title of the panel shows which step is to be defined. Check the box in front of the chosen step and click ‘Continue’. When the last step of the divergence is defined, the construction of the common reduct for the given divergence will appear on the screen.

3.5 Tips and tricks

The user interface that comes with the program was created to simplify the use of the program. However, in case of frequent use, there are a couple of useful tips and tricks, that involve ways to get round the GUI. To do this, one needs to change or add some parts to the source code of the program. We will explain how this can be done below.

First of all, the GUI is a fine way to add new trs’s to the program, in case of sporadic use of the program, but this can be quite exhaustive when this new trs, for some reason, needs to be available for use more often. As it happens, creating a new trs requires the user to define sets of functions, variables and rules. These required features can be entered into the textfields in the GUI by the user. But once the program is closed down, these additions will be lost the next time the program is executed. However, it is possible to add new trs’s to the program in a lasting way, by adding them to the source code itself in the class ‘Startmodel’. There is a standard couple of trs’s that are created at the startup of the program. The codes that create these trs’s can be followed to add more standard trs’s to the program. In some cases this might be more practical than using the GUI.

Another characteristic of the GUI is that the shown output, the description of the constructed confluence, only entails the outside reductions of the diagram. The recursive steps that come in between and the inner tile solutions are left out. If one would like to be able to also look at these intermediary steps, this is possible by setting the values of (some of) the printing booleans at the top of the classes ConfluenceModular, Mix, TallTall and ShortShort, to ‘true’. All intermediate steps will then be printed to the console.

4 Nawoord

Het schrijven van dit programma, was voor mij een bevestiging dat ik programmeren erg leuk vind. Het hebben van een duidelijk en concreet doel, werkte goed voor me, omdat ik daarbij meer houvast had, dan bij het schrijven van een bewijs voor een stelling. De theorie van het termherschrijven bleek, nadat ik de tijd had genomen me er rustig in te verdiepen, minder ontoegankelijk dan deze in eerste instantie leek. Ik heb veel gehad aan de uitleg van Vincent tijdens onze bijeenkomsten en aan de colleges van Roel de Vrijer, die ik aan de VU gevolgd heb.

Nu mijn stage erop zit en mijn scriptie af is, kijk ik met veel plezier terug op het afgelopen jaar. Ik vond het uiteindelijk echt leuk om te doen, en mijn samenwerking met Vincent is ook goed bevallen. Ik ben heel blij met de keuzes die we (Vincent en ik) gemaakt hebben.

Het programma zal gebruikt gaan worden door Vincent ter ondersteuning van zijn theorie tijdens presentaties. Zoals eerder uitgelegd, is het mogelijk de stelling te gebruiken om aan te tonen dat een softwareprogramma een bepaalde eigenschap heeft en dat kan met 'Confluence Constructor' constructief worden bewezen. Tevens zou het programma gebruikt kunnen worden voor het vinden van boven- en ondergrenzen voor combinaties van ordeningen, zoals bijvoorbeeld de grootste gemene deler of het kleinste gemene product. Als laatste is het wellicht interessant dit programma te gebruiken om empirische data te verzamelen over de bovengrenzen op de lengtes van de reducties die nodig zijn om het gemeenschappelijk reduct te vinden.

Referenties

[1] F. Baader and T. Nipkow, Term Rewriting and All That. *Cambridge, England: Cambridge University Press. (1999)*

[2] J.A. Bergstra, J.W. Klop, A. Middeldorp. Termherschrijfsystemen. *Amsterdam: Kluwer Bedrijfswetenschappen. (1989)*

[3] M. Bezem, J.W. Klop, R. Vrijer, Terese (Group). Term Rewriting Systems. *Cambridge, England: Cambridge University Press. (2003)*

[4] V. van Oostrom. Confluence by Decreasing Diagrams, Converted. *Proceedings of the 19th RTA (RTA 2008), Hagenberg, pp. 306-320. (2008, July)*

[5] V. van Oostrom. Modularity of Confluence, Constructed. *Proceedings of the 4th IJCAR (IJCAR 2008), Sydney, pp. 348-363. (2008, August)*

[6] Y. Toyama. On the Curch-Rosser property for the direct sum of term rewriting systems.. *Journal of the Association for Computing Machinery, 34(1), 128-143. (1987)*

[7] H. Schildt. Java: A Beginner's Guide. *Emeryville, California: McGraw-Hill/Osborne. (2005)*