GETALLEN EN FUNCTIES IN DE π-CALCULUS

Een verkennende aanzet

10 mei 2007

Met dank aan Henk Barendregt en Jan-Willem Klop

Freek Verbeek

Radboud Universiteit, Nijmegen

Inhoudsopgave

1 Inleiding

Er zijn een paar verschillende soorten procesalgebra die allemaal op verschillende manieren processen beschrijven. Ze hebben zo hun voor- en nadelen en onderscheiden zich onder andere op expressiekracht en elegantie. De π-calculus is een procesalgebra waarin processen beschreven worden als een geheel van allerlei verschillende onderling communicerende onderdelen. In deze scriptie wil ik onderzoek doen naar de π-calculus en haar mogelijkheden.

Een groot voordeel van de π-calculus ten opzichte van andere procescalculi is de mogelijkheid om mobiliteit te modelleren. Dit wil zeggen: het verplaatsen van links (communicatiekanalen) tussen processen. Mobiliteit is een belangrijk issue in de hedendaagse Informatica; praktische voorbeelden zijn processen in complexe netwerken, of protocollen tussen mobiele telefoons. Een degelijk model is interessant omdat het het een basis vormt voor het oplossen van veel security-gerelateerde problemen of het valideren van protocollen in dergelijke mobiele systemen.

Het is interessant om uit te zoeken wat de expressiekracht is van de π-calculus. De taal die wordt gebruikt om processen te definiëren is zeer eenvoudig, zoals zal blijken in hoofdstuk 2.1. Toch is de π-calculus een universeel Turingcompleet model. Dit is ooit bewezen door de π-calculus te encoderen in de λ-calculus [Milner].

Hiermee is niet alleen bewezen dat de π-calculus het concept van getallen kan modelleren, maar ook alle Turingberekenbare functies. In deze bachelorscriptie zal ik onderzoeken hoe dit gedaan moet worden. Daarmee zal ik dit op een andere, constructievere, manier de Turingvolledigheid van de π-calculus bewijzen.

2 π-calculus

Dit hoofdstuk leidt kort de π-calculus in zoals hij in deze bachelorscriptie gebruikt gaat worden.

2.1 Standaard π-calculus

De π-calculus is de verzameling processen P^π die als volgt wordt gedefinieerd:

$$\pi := \overline{x}\langle y\rangle \mid x(z) \mid \tau$$
$$P := \sum_{i\in I} \pi_i.P_i \mid P \mid P' \mid (\boldsymbol{\nu} z)(P) \mid !P$$

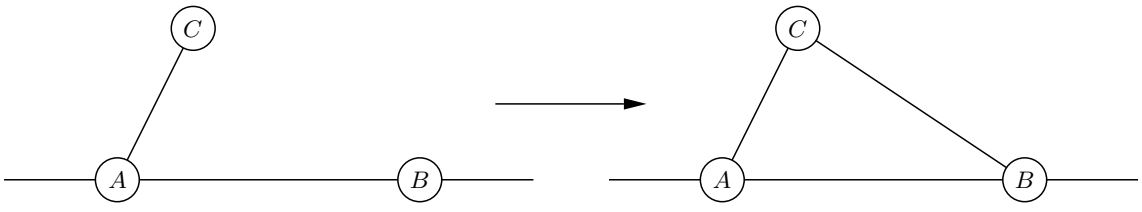
Hierbij is π de verzameling actieprefixen, die respectievelijk staan voor een zendactie, een ontvangactie en een abstracte actie.

Deze definitie van de π-calculus komt uit [Milner99]. Het gebruik van $\boldsymbol{\nu}$ in plaats van **new** is afkomstig van [SW01]. De mogelijkheid tot sequentiële compositie is niet direct in de definitie meegnomen. Dit is echter gemakkelijk te simuleren, zie hiervoor hoofdstuk 9 van [Milner99]. De prefix $[x == y]\pi$ uit [SW01], die de gelijkheid van twee kanalen test, wordt niet gebruikt.

De variabelen x, y en z worden names genoemd. Names zijn de namen van de kanalen waarover een actie kan gebeuren. De acties $x(z).P$ en $(\boldsymbol{\nu} z)(P)$ binden de name z aan scope P. Een gebonden name kan niet buiten zijn scope worden gebruikt. Een name is free als hij niet gebonden is in dat proces.

2.2 Mobiliteit

De π-calculus biedt mogelijkheid om processen met mobiliteit te modelleren. Een minimaal voorbeeld van mobiliteit is de overgang van situatie 1 van figuur 2.1 naar situatie 2. In situatie 1 kan proces A communiceren met proces B en proces C niet. De π-calculus bevat reductiemogelijkheden om het kanaal tussen A en B te communiceren naar proces C. Dit betekent dat, indien op de juiste manier gemodelleerd, kanalen, volledige processen en data mobiel kunnen zijn en van proces naar proces kunnen gaan [Wikipedia].



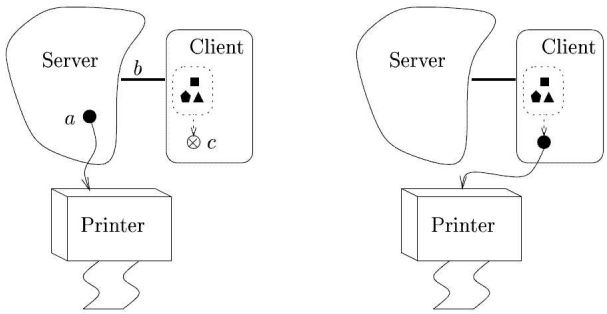
Situatie 1 Situatie 2

Figuur 2.1. Mobiliteit

Concrete voorbeelden van mobiliteit zijn genoeg te vinden in de hedendaagse wereld. Bijvoorbeeld het gebruik van mobiele telefoons en geavanceerde netwerkapplicaties. Figuur 2.2 is een voorbeeld van mobiliteit in een client-server setting. Figuur 2.3 is een ander voorbeeld. Hierin wordt het proces gemodelleerd van een auto die communiceert met een centrum via de dichtstbijzijnde basis.

2.3 Polyadische π-calculus

In de standaard π-calculus biedt een output actie $\overline{x}\langle y\rangle$ slechts de mogelijkheid één name tegelijk te versturen. Voor bijna alle toepassingen van de π-calculus is het echter handig om over de mogelijkheid te beschikken om via één output actie meerdere names tegelijk te kunnen versturen. Dit



Figuur 2.2. Mobiliteit in een netwerk [Parrow]

is mogelijk met de polyadische π-calculus. In deze versie van de π-calculus kan de actie prefix $\overline{x}\langle y\rangle$ uitgebreid worden tot $\overline{x}\langle y_1, y_2, \ldots, y_k\rangle$ en de actieprefix $x(y)$ analoog. Milner heeft in [Milner99] laten zien dat de polyadische π-calculus te simuleren is met de standaard π-calculus en dus zal in deze scriptie van deze mogelijkheid gebruik gemaakt worden wanneer nodig.

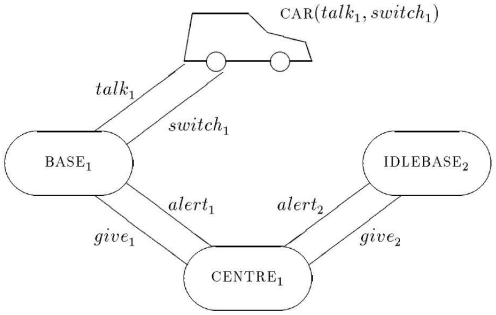
2.4 Parametrische processen

Het is mogelijk een proces te definiëren dat een of meerdere parameters verwacht. Een proces P dat parameters $n_1, n_2, \ldots, n_k$ verwacht, wordt geschreven als $P\langle n_1, n_2, \ldots, n_k\rangle$. Een voorbeeld van een parametrisch proces is proces 2.1 uit [Milner99], dat een buffer voorstelt:

$$\begin{aligned} B\langle l, r\rangle &= l(x).C\langle x, l, r\rangle \\ C\langle x, l, r\rangle &= \overline{r}\langle x\rangle.B\langle l, r\rangle \end{aligned} \tag{2.1}$$

2.5 Structurele congruentieregels

De regels uit tabel 2.1 geven structurele congruentie aan en komen uit [SW01]. De regel Sc-rep geeft de betekenis van replicatie: er kunnen oneindig veel identieke kopieën van het proces P worden gegenereerd. De regel Sc-res-rep is belangrijk bij het reduceren van de restrictieprefix. De regel kan twee kanten op toegepast worden en kan dus gebruikt worden om een scope uit te breiden en om hem te verkleinen.



Figuur 2.3. Mobiliteit in communicatie [Milner91]

SC-SUM-ASSOC	$P_1 + (P_2 + P_3)$	$\equiv$	$(P_1 + P_2) + P_3$
SC-SUM-ASSOC	$P_1 + P_2$	$\equiv$	$P_2 + P_1$
SC-SUM-INACT	$P + \mathbf{0}$	$\equiv$	P
SC-COMP-ASSOC	$P_1 \mid (P_2 \mid P_3)$	$\equiv$	$(P_1 \mid P_2) \mid P_3$
SC-COMP-COMM	$P_1 \mid P_2$	$\equiv$	$P_2 \mid P_1$
SC-COMP-INACT	$P \mid \mathbf{0}$	$\equiv$	P
SC-RES	$\boldsymbol{\nu} z \boldsymbol{\nu} w P$	$\equiv$	$\boldsymbol{\nu} w \boldsymbol{\nu} z P$
SC-RES-INACT	$\boldsymbol{\nu} z \mathbf{0}$	$\equiv$	$\mathbf{0}$
SC-RES-REP	$\boldsymbol{\nu} z (P_1 \mid P_2)$	$\equiv$	$P_1 \mid \boldsymbol{\nu} z P_2$, if $z \notin names(P_1)$
SC-REP	$!P$	$\equiv$	$P \mid !P$

Tabel 2.1. Axiomas voor structurele congruentie

2.6 Reductie regels

Zie tabel 2.2. De regel R-INTER is de regel die de essentie van mobiliteit modelleert. Hij is aangepast voor de polyadische π-calculus. De regel R-TAU wordt niet gebruikt in deze scriptie. τ wordt alleen gebruikt om representaties te maken, maar is nooit het resultaat van een reductie.

R-INTER	$\overline{(\overline{x}\langle \overline{y} \rangle.P_1 + M_1) \mid (x(\overline{z}).P_2 + M_2) \rightarrow P_1 \mid P_2[\overline{z} = \overline{y}]}$
R-TAU	$\overline{\tau.P + M \rightarrow P}$
R-PAR	$\dfrac{P_1 \rightarrow P_1'}{P_1 \mid P_2 \rightarrow P_1' \mid P_2}$
R-RES	$\dfrac{P \rightarrow P'}{\boldsymbol{\nu} z P \rightarrow \boldsymbol{\nu} z P'}$
R-STRUCT	$\dfrac{P \rightarrow P'}{Q \rightarrow Q'}$ als $P \equiv Q$ en $P' \equiv Q'$

Tabel 2.2. Reductie regels

3 Getallen

3.1 Introductie

Wat de π-calculus onderscheidt van andere procesalgebra als ACP_τ is de mogelijkheid om mobiliteit te modelleren. Het is de vraag of het mogelijk is getallen en functies op een dergelijke manier te modelleren dat ze mobiel blijven en dus over een kanaal gecommuniceerd kunnen worden naar een ander proces. Ik zal onderzoeken of er procesgrafen zijn voor ieder getal $n \in \mathbb{N}$, het versturen en ontvangen van dat getal en het binden van het getal aan een proces. Hetzelfde ook voor iedere functie $f : \mathbb{N} \mapsto \mathbb{N}$.

De functie $\mathbb{G}$ wordt ingevoerd, die van een bepaald wiskundig object (in dit geval een getal of een functie) de procesgraaf in de π-calculus geeft. Deze functie zal dus gedefinieerd worden in de volgende hoofdstukken.

Verder zal de functie $names(P)$ gebruikt worden, die resulteert in een verzameling van alle names die in het proces P gebruikt worden. In [SW01] wordt deze functie $n(P)$ genoemd, maar voor de overzichtelijkheid geef ik hem hier een duidelijkere naam.

De getallen die in deze scriptie worden gebruikt zijn, zolang niet anders vermeld, natuurlijke getallen. De namen waarmee ze worden aangeduid zijn over het algemeen n of m. Functies krijgen over het algemeen de naam f en names van communicatiekanalen zijn c, d, of e.

3.2 Representatie

Milner representeert in [Milner91] getallen voor de π-calculus.

Definitie 3.1. *Laat $n \in \mathbb{N}$. De representatie van n op kanalen xz, notatie $\mathbb{G}[n_{xz}]$, is*

$$\mathbb{G}[n_{xz}] \stackrel{def}{=} (\overline{x}.)^n \overline{z}$$

Milner gebruikt een andere notatie voor $\mathbb{G}[n_{xz}]$, namelijk: $\underline{n}(xz)$.

Een getal wordt dus gedefinieerd met behulp van twee kanalen: een kanaal x dat de waarde van het getal weergeeft door n keer een zendactie uit te voeren op dat kanaal. Het is niet nodig daadwerkelijk een name over het kanaal te zenden; bij het gebruik van de polyadische π-calculus is het toegestaan een lege vector te verzenden. Het tweede kanaal z zorgt dat gecontroleerd kan worden of het getal gelijk is aan nul. Zo heeft het proces $\overline{z}$ waarde nul en proces $\overline{xxz}$ waarde twee. In de praktijk zal aan de kanalen x en z een subscript toegevoegd worden (bijvoorbeeld de naam van de variabele die naar dit getal verwijst) zodat de kanalen een unieke naam hebben ten opzichte van de rest van het proces.

Een getal "lezen" wordt gedaan door op het x kanaal van het getal te ontvangen. Testen of het getal gelijk is aan nul kan door te ontvangen over het kanaal z. Daarom heeft deze definitie een groot nadeel: de getallen kunnen maar één keer gebruikt worden. Zodra ze gelezen worden, bestaan ze niet meer. Het is mogelijk de definitie uit te breiden om dit te voorkomen, maar in deze scriptie heb ik aan definitie 3.1 genoeg.

3.3 Restrictie voor getallen

Een getal bestaat dus theoretisch gezien uit twee names die dus ergens met de restrictieprefix ($\boldsymbol{\nu}$) gebonden kunnen worden. Het binden van een getal n gaat dus door middel van het binden van de twee kanalen die in het proces $\mathbb{G}[n]$ voorkomen.

Definitie 3.2. *Laat $n \in \mathbb{N}$ zijn gerepresenteerd op kanalen xz. De representatie van restrictie van n, notatie $(\boldsymbol{\nu} n)$, is*

$$(\boldsymbol{\nu} n) \stackrel{def}{=} (\boldsymbol{\nu} xz)$$

Proces 3.1 geeft een voorbeeld van een proces dat n keer een actie doet (in dit geval, zenden op een kanaal c) en vervolgens één keer een zendactie doet op kanaal d.

$$\begin{aligned} P &:= (\boldsymbol{\nu} n)(\mathbb{G}[n_{xz}] \mid P') \\ P' &:= x.\overline{c}.P' + z.\overline{d} \end{aligned} \tag{3.1}$$

Stel dat in bovenstaand voorbeeld $n = 4$, dan is proces 3.1 gelijk aan proces 3.2:

$$\begin{aligned} P &:= (\boldsymbol{\nu} x, z)(\overline{xxxxz} \mid P') \\ P' &:= x.\overline{c}.P' + z.\overline{d} \end{aligned} \tag{3.2}$$

Proces 3.2 reduceert naar $\overline{ccccd}$.

3.4 Mobiliteit van getallen

Het versturen van een getal houdt niet meer in dan het versturen van de names x en z die het proces van het getal gebruikt. Hetzelfde geldt voor het ontvangen van een getal. Formeel uitgedrukt:

Definitie 3.3. *Laat $n \in \mathbb{N}$ zijn gerepresenteerd op kanalen xz. De representatie van versturen en verzenden van n op kanaal c, notatie $\overline{c}\langle z \rangle$ en $c(n)$, is*

$$\begin{aligned} \overline{c}\langle n \rangle &\stackrel{def}{=} \overline{c}\langle xz \rangle \iff n \in \mathbb{N} \\ c(n) &\stackrel{def}{=} c(xz) \iff n \in \mathbb{N} \end{aligned}$$

Proces 3.3 geeft een voorbeeld van mobiliteit van een getal:

$$\begin{aligned} P \quad &:= \quad (\boldsymbol{\nu} 3)(A \mid B) \mid C \\ &\text{met} \quad A = \overline{c}\langle 3 \rangle.A' \\ &\text{met} \quad B = \mathbb{G}[3] \\ &\text{met} \quad C = c(k).C' \end{aligned} \tag{3.3}$$

Volledig uitgeschreven in de π-calculus komt dit neer op 3.4:

$$\begin{aligned} P \quad &:= \quad (\boldsymbol{\nu} x, z)(A \mid B) \mid C \\ &\text{met} \quad A = \overline{c}\langle x, z \rangle.A' \\ &\text{met} \quad B = \overline{x}.\overline{x}.\overline{x}.\overline{z} \\ &\text{met} \quad C = c(x_k, z_k).C' \end{aligned} \tag{3.4}$$

4 Functies

Het is mogelijk in de π-calculus wiskundige functies te definiëren. Ik definieer in eerste instantie alleen functies van de vorm $f : \mathbb{N} \mapsto \mathbb{N}$. Een proces dat een dergelijke functie uitvoert, zal zijn invoer lezen van de poorten x_i en z_i en versturen naar de poorten x_o en z_o.
Als voorbeeld definieer ik de functie $f_3 : \mathbb{N} \mapsto \mathbb{N}$, met $f_3(n) = n+3$. Het proces zendt het resultaat naar m. Het voorbeeld en het gebruik van de functie *Copy* is afkomstig uit [Milner91].

$$\mathbb{G}[f_3] = \overline{x}_o.\overline{x}_o.\overline{x}_o.Copy(x_i z_i, x_o z_o) \tag{4.1}$$

Proces 4.1 loopt dan tegelijkertijd met het proces dat het getal n representeert.

Definitie 4.1. *Een functie $f : \mathbb{N} \mapsto \mathbb{N}$ is representeerbaar via $x_i z_i$ en $x_o z_o$ door $\mathbb{G}[f]$ als geldt:*

$$\mathbb{G}[n_{x_i z_i}] \mid \mathbb{G}[f] \twoheadrightarrow \mathbb{G}[m_{x_o z_o}] \Longleftrightarrow f(n) = m$$

Als in het proces in definitie 4.1 de functie f_3 wordt toegepast, wordt (niet noodzakelijk in deze volgorde) het volgende uitgevoerd:

- Er wordt n keer verzonden over x_i en één keer over z_i, dit is het 'maken' van het getal n ($\mathbb{G}[n_{x_i z_i}]$);
- er wordt drie keer verzonden over het resultaatkanaal x_o, dit is het geven van de waarde drie aan m;
- er wordt n keer over x_o verzonden, zodat daarover in totaal $n + 3$ keer is verzonden (*Copy*);
- m wordt afgemaakt door het ontvangen over z_i en het vervolgens verzenden van z_o (*Copy*)

Het resultaat is dus het proces $(\overline{x_o}.)^{n+3}.\overline{z_o}$, per definitie gelijk aan $\mathbb{G}[m_{x_o z_o}]$, met $m = n + 3$. Oftewel, m heeft de waarde $n + 3$. Een formeel bewijs is te vinden in [Milner91].

Een andere manier om functies te definiëren is is als volgt:

$$\mathbb{G}[n_{x_i z_i}] \mid \mathbb{G}[f] \twoheadrightarrow \mathbb{G}[f(n)_{x_o z_o}] \tag{4.2}$$

Definitie 4.1 en 4.2 zijn echter alleen gelijk als de π-calculus de eigenschap uniciteit heeft. Dit is nog een open vraag. De π-calculus is niet confluent, maar het is mogelijk dat op een andere manier uniciteit bewezen kan worden.

4.1 Restrictie en mobiliteit van functies

Nadelen van het representeren van een functie zoals dat is gedaan in definitie 4.1 is dat het niet mogelijk is naar de functie te verwijzen en dat de functie kortstondig is. Het is daarom ook niet mogelijk de functie als object te versturen naar andere processen of om de restrictie operator er op toe te passen.

Het is mogelijk de representatie $\mathbb{G}[f]$ op een dergelijke manier toe te passen dat deze nadelen niet meer van toepassing zijn, door de representatie uit te breiden met de representatie van een functieaanroep en de representatie van de code van de functie.

Definitie 4.2. *Laat $f : \mathbb{N} \mapsto \mathbb{N}$ representeerbaar zijn door $\mathbb{G}[f]$. De representatie van de functie-aanroep van f op kanaal c_f is*

$$\mathbb{G}_2[f]\langle x_i z_i, x_o z_o\rangle \stackrel{def}{=} \overline{c_f}\langle x_i z_i, x_o z_o\rangle$$

De representatie van de code van f op kanaal c_f is

$$\mathbb{G}_1[f] \stackrel{def}{=} c_f(x_i z_i, x_o z_o).(\mathbb{G}[f])$$

Proces $\mathbb{G}_2[f]$ is geparametriseerd met de kanalen van de invoer (x_i en z_i) en van de uitvoer (x_o en z_o). Er wordt een uniek kanaal gebruikt, dat voor functie f de naam c_f zal krijgen, dat de "naam" van de functie wordt voor de omgeving van het proces. Deze naam kan worden gebruikt om naar de functie te verwijzen en dus om er restrictie op toe te passen, om hem te versturen of om hem te ontvangen.

Het is onmogelijk recursieve functies te representeren als de functies alleen kortstondig representeerbaar zijn. Het gebruik van $\mathbb{G}_1$ en $\mathbb{G}_2$ maakt het mogelijk een functie meerdere keren aan te roepen. Het maakt daarom ook recursieve functies mogelijk. Dit wordt weergegeven in het volgende lemma.

Lemma 4.3. *Laat $f : \mathbb{N} \mapsto \mathbb{N}$ representeerbaar zijn door $\mathbb{G}[f]$. Dan geldt*

$$\mathbb{G}[n_{x_i z_i}] \mid !\mathbb{G}_1[f] \mid \mathbb{G}_2[f]\langle x_i z_i, x_o z_o\rangle \twoheadrightarrow !\mathbb{G}_1[f] \mid \mathbb{G}[m_{x_o z_o}] \Longleftrightarrow f(n) = m$$

Zoals vermeld, maken $\mathbb{G}_1$ en $\mathbb{G}_2$ het mogelijk om naar de functie f te verwijzen, door middel van c_f. Dit betekend dat restrictie, het verzenden en het ontvangen van een functie triviaal zijn geworden, maar voor de volledigheid geef ik hier de definities.

Definitie 4.4. *Laat $f : \mathbb{N} \mapsto \mathbb{N}$ representeerbaar zijn door $\mathbb{G}[f]$ en laat de functieaanroep en de functiecode zijn gerepresenteerd op kanaal c_f.*
De representaties van restrictie van f en het versturen en verzonden van f op kanaal c, notatie $(\boldsymbol{\nu} f)$, $\overline{c}\langle f\rangle$ en $c(f)$, zijn dan

$$(\boldsymbol{\nu} f) \stackrel{def}{=} (\boldsymbol{\nu} c_f)$$
$$\overline{c}\langle f\rangle \stackrel{def}{=} \overline{c}\langle c_f\rangle$$
$$c(f) \stackrel{def}{=} c(c_f)$$

Het volgende proces geeft een voorbeeld van een proces dat mobiliteit van functies gebruikt. Proces A verstuurt de functie f, met $f(n) = n + 3$ naar proces C, dat vervolgens die functie toepast op het getal 5 en vervolgens het resultaat terugstuurt naar A. Aan de rechterkant staat het proces volledig uitgeschreven.

P	$:= (\boldsymbol{\nu} f)(A \mid B) \mid (\boldsymbol{\nu} 5)(C \mid D)$	$\stackrel{\text{def}}{=} (\boldsymbol{\nu} c_f)(A \mid B) \mid (\boldsymbol{\nu} x_5 z_5)(C \mid D)$
met		met
A	$:= \overline{d}\langle f\rangle.d(n_{xz})$	$\overline{d}\langle c_f\rangle.d(x_n z_n) \mid A'$
B	$:= \mathbb{G}_1[f]$	$c_f(x_n z_n, x_m z_m).(\overline{x_m x_m x_m}.Copy(x_n z_n, x_m z_m))$
C	$:= d(f).\mathbb{G}_2[f]\langle x_5 z_5, x_m z_m\rangle.\overline{d}\langle m_{x_m z_m}\rangle$	$c(c_f).(\boldsymbol{\nu} x_m z_m)(\overline{c_f}\langle x_5 z_5, x_m z_m\rangle.\overline{d}\langle x_m z_m\rangle)$
D	$:= \mathbb{G}[5]$	$\overline{x_5 x_5 x_5 x_5 x_5 z_5}$

Uiteindelijk reduceert dit proces naar $\mathbb{G}[n_{xz}]$, met $n = 8$.

5 μ-recursieve functies

5.1 Inleiding

Er is nu aangetoond dat alle getallen mobiel gemodelleerd kunnen worden en er is hiervoor een gemakkelijke notatie, in de vorm van $\overline{x}\langle n\rangle$ en $x(n)$ ingevoerd. Ook is een aanzet gegeven hoe functies gemodelleerd moeten worden in de π-calculus en hoe simpele arithmetiek plaatsvindt. In dit hoofdstuk zal worden aangetoond dat de π-calculus krachtig genoeg is om de μ-recursieve functies te modelleren. Om dit te bewijzen, moeten de initiële functies gerepresenteerd worden en moet vervolgens worden aangetoond dat de π-definieerbare functies gesloten zijn onder compositie, primitieve recursie en minimalisatie [Barendregt].

5.2 Gebruikte notatie

In dit hoofdstuk zal de volgende notatie gebruikt worden:

$\overrightarrow{n}$	Vanaf nu zal voor een rij getallen $n_1, n_2, \ldots, n_k$ de notatie $\overrightarrow{n}$ worden gebruikt. Aangezien de polyadische π-calculus wordt gebruikt, kan een dergelijke rij op dezelfde manier worden behandeld als een normale name. Laat x_n en z_n de kanalen zijn van de rij $\overrightarrow{n}$, dan zijn x_{n_i} en z_{n_i} de kanalen van het i-de element.
$\varphi(\overrightarrow{n})$	In dit hoofdstuk worden niet alleen unaire functies van de vorm $f : \mathbb{N} \mapsto \mathbb{N}$ gerepresenteerd, maar ook k-aire functies van de vorm $\varphi : \mathbb{N}^k \mapsto \mathbb{N}$.
$\mathbf{n^c}$	Zoals vermeld in hoofdstuk 3.2 zijn getallen kortstondig. In onderstaande definities is het soms nodig getallen te kopiëren en de kopie te gebruiken. Om niet te verzanden in dergelijke details, introduceer ik de superscript ^{c}. Het proces n^c staat voor een kopie van n ($n \in \mathbb{N}$).

5.3 Initiële functies

De functie $U_l^k(\overrightarrow{n}) = n_l (0 \leq l < k)$ wordt gerepresenteerd via $x_i z_i$ en $x_o z_o$door:

$$\mathbb{G}[U_l^k] \stackrel{\text{def}}{=} Copy(x_{i_l} z_{i_l}, x_o z_o) \tag{5.1}$$

In de representatie van $U_l^k(\overrightarrow{n})$ komt k niet voor. Bij de definitie van een representatie van een functie, wordt er stilzwijgend vanuit gegaan dat de input op kanalen $x_i z_i$ staat. Als gevolg van de gebruikte notatie in dit hoofdstuk, betekent dit dat, wanneer de input een rij is van grootte k, dat dan de kanalen van de elementen van die rij staan op de kanalen $x_{i_n} z_{i_n}$, met $0 \leq n < k$. Zie als voorbeeld het proces 5.2, dat reduceert naar $\mathbb{G}[m_{x_o z_o}]$ dan en slechts dan als $m = n_1$.

$$P = \mathbb{G}[\overrightarrow{n}_{x_i z_i}] \mid \mathbb{G}[U_1^3] \tag{5.2}$$

De functie $S^+(n) = n + 1$ wordt gerepresenteerd via $x_i z_i$ en $x_o z_o$ door:

$$\mathbb{G}[S^+] \stackrel{\text{def}}{=} \overline{x_o}.Copy(x_i z_i, x_o z_o) \tag{5.3}$$

De functie $Z(n) = 0$ wordt gerepresenteerd via $x_i z_i$ en $x_o z_o$ door:

$$\mathbb{G}[Z] \stackrel{\text{def}}{=} \overline{z_o} \tag{5.4}$$

5.4 Compositie

Laat χ, ψ_1, ψ_2 zijn gerepresenteerd door $\mathbb{G}[\chi]$, $\mathbb{G}[\psi_1]$ en $\mathbb{G}[\psi_2]$. Dan wordt de functie

$$\varphi(\overrightarrow{n}) = \chi(\psi_1(\overrightarrow{n}), \psi_2(\overrightarrow{n})) \tag{5.5}$$

gerepresenteerd door

$$\begin{aligned}\mathbb{G}[\varphi] \stackrel{\text{def}}{=} \mathbb{G}_1[\chi] \mid \mathbb{G}_1[\psi_1] \mid \mathbb{G}_1[\psi_2] \mid (\boldsymbol{\nu}\,\overrightarrow{r}) \\ (\mathbb{G}_2[\psi_1]\langle x_i z_i, x_{r_1} z_{r_1}\rangle.\mathbb{G}_2[\psi_2]\langle x_i z_i, x_{r_2} z_{r_2}\rangle.\mathbb{G}_2[\chi]\langle x_r z_r, x_o z_o\rangle)\end{aligned} \tag{5.6}$$

De ψ-functies worden allen uitgevoerd. De resultaten komen in een rij $\overrightarrow{r}$. Vervolgens wordt χ uitgevoerd met de resultaatrij $\overrightarrow{r}$. Het resultaat hiervan komt in m. Zonder verlies van algemeenheid wordt hier als voorbeeld de representatie van φ met twee parameters gegeven.

5.5 Primitieve recursie

Laat χ en ψ zijn gerepresenteerd door $\mathbb{G}[\chi]$ en $\mathbb{G}[\psi]$. Dan wordt de functie

$$\begin{aligned}\varphi(0, \overrightarrow{n}) &\stackrel{\text{def}}{=} \chi(\overrightarrow{n}) \\ \varphi(k+1, \overrightarrow{n}) &\stackrel{\text{def}}{=} \psi(\varphi(k, \overrightarrow{n}), k, \overrightarrow{n})\end{aligned} \tag{5.7}$$

gerepresenteerd door

$$\begin{aligned}\mathbb{G}[\varphi] \stackrel{\text{def}}{=} \mathbb{G}_1[\chi] \mid \mathbb{G}_1[\psi] \mid (\boldsymbol{\nu} r_\varphi)(\overline{r_\varphi}\langle k_{x_k z_k}, n_{x_i z_i}, m_{x_o z_o}\rangle \mid !r_\varphi(k_{x_k z_k}, n_{x_i z_i}, m_{x_o z_o}).P) \\ P = z_k.\mathbb{G}_2[\chi]\langle x_i z_i, x_o z_o\rangle + x_k.(\boldsymbol{\nu} p)(\overline{r_\varphi}\langle k^c_{x_{k'} z_{k'}}, n^c_{x_{i'} z_{i'}}, p_{x_p z_p}\rangle.\mathbb{G}_2[\psi]\langle x_p z_p, x_{k'} z_{k'}, x_{i'} z_{i'}, x_o z_o\rangle)\end{aligned} \tag{5.8}$$

Recursie kan binnen de π-calculus bereikt worden met replicatie [Milner99]. Er wordt een nieuw kanaal r_φ aangemaakt. Hierover data versturen betekent een keer de recursie ingaan met de verstuurde data als parameters. Het proces $!r_\varphi(\ldots).P$ kan gezien worden als de code van de recursieve procedure P. In pseudocode staat hier:

```
function P(int k, int[] n):
    if k = 0 then
        return χ(n);
    else
        k := k − 1;
        int p;
        p := P(k, n);
        return ψ(p, k, n)
.
```

5.6 Minimalisatie

Laat χ en S^+ gerepresenteerd zijn door $\mathbb{G}[\chi]$ en $\mathbb{G}[S^+]$. Dan wordt de functie

$$\varphi(\overrightarrow{n}) = \mu k[\chi(\overrightarrow{n}, k) = 0] \tag{5.9}$$

gerepresenteerd door

$$\begin{aligned}\mathbb{G}[\varphi] \stackrel{\text{def}}{=} \mathbb{G}_1[\chi] \mid (\boldsymbol{\nu} r_\varphi) \\ (\overline{z_0} \mid \overline{r_\varphi}\langle \overrightarrow{n}_{x_i z_i}, 0_{x_0 z_0}, m_{x_o z_o}\rangle \mid !r_\varphi(\overrightarrow{n}_{x_i z_i}, k_{x_k z_k}, m_{x_o z_o}).P) \\ P = (\boldsymbol{\nu} p)(\mathbb{G}_2[\chi]\langle x_i z_i, x_k z_k, x_p z_p\rangle. \\ (z_p.Copy(x_k z_k, x_o z_o) + x_p.\mathbb{G}_2[S^+]\langle x_k z_k, x_{k'} z_{k'}\rangle.\overline{r_\varphi}\langle \overrightarrow{n}_{x_i z_i}, k'_{x_{k'} z_{k'}}, m_{x_o z_o}\rangle)\end{aligned} \tag{5.10}$$

```
function P(int[] n, int k):
    int p;
    p := χ(n,k);
    if p = 0 then
        return k;
    else
        k' := k + 1;
        return P(n, k')
.
```

Ook om de minimalisatiefunctie te definiëren wordt gebruik van recursie gemaakt. De recursieve procedure P doet het volgende in pseudocode:

Aanvankelijk wordt deze functie aangeroepen met $k = 0$.

6 Conclusie

Op een informele wijze heb ik aangetoond dat de π-calculus Turingcompleet is. Dit is gedaan door middel van het representeren van Kleene's μ-recursieve functies. Toch zijn er nog veel open vragen en zijn er genoeg punten die een grondigere en meer formele aanpak verdienen. Aangezien dit slechts een bachelorscriptie is, is daar echter te weinig tijd voor en moet deze scriptie vooral gezien worden als aanzet tot een bewijs. Hier een overzicht van punten die beter en grondiger aangepakt zouden moet worden om het bewijs overtuigend te maken:

Uniciteit	Het is nog een open vraag of de π-calculus uniciteit heeft.
n-aire functies	In deze scriptie worden unaire functies gerepresenteerd. De representatie van n-aire functies moet ook uitgewerkt worden.
Notationele en grafische presentatie	Om bij het representeren van getallen en functies in de π-calculus niet te verzanden in een notationeel moeras, moet er een betere notatie uitgewerkt worden. Ook is het praktisch een grafische representatie uit te werken.
Pseudocodes	De relatie tussen de pseudocodes van hoofdstuk 5 en de bijbehorende π-termen moet meer uitgewerkt worden.
Lemma's	De lemma's en proposities moeten formeel bewezen worden.
Onderzoek naar tools	Het gebruik van tools die uitgebreide voorbeelden aankunnen kan de stellingen ondersteunen.
Kortstondigheid	Het feit dat functies en getallen kortstondig zijn en recursie niet kan worden bereikt zonder replicatie moet duidelijker worden uitgewerkt.
Codering λ-calculus	Het bewijs wat hier geleverd wordt kan ook gedaan worden door de λ-calculus te representeren in de π-calculus. Het is echter onduidelijk of dit volledig kan en dit moet verder uitgezocht worden.

Referenties

[Milner] Robin Milner, *Functions as Processes*, verschenen in *Mathematical Structures in Computer Science, Vol. 2* (1992)

[Milner99] Robin Milner, *Communicating and mobile systems: the π-calculus* (1999)

[SW01] Davide Sangiorgi & David Walker, *The π-calculus, A theory of mobile processes* (2001)

[Milner91] Robin Milner, *The Polyadic π-calculus: a tutorial* (1991)

[Parrow] Joachim Parrow, *An introduction to the π-calculus*, verschenen in *Handbook of Process Algebra* (Bergstra, Ponse, Smolka, 2001)

[Barendregt] Henk Barendregt & Erik Barendsen, *Introduction to Lambda Calculus*, (2000)

[Wikipedia] *Wikipedia, The Free Encyclopedia* (2006)