Funktionale Berechnung in einem uniform nebenläufigen Kalkül mit logischen Variablen

Dissertation
zur Erlangung des Grades
Doktor der Naturwissenschaften
der Technischen Fakultät
der Universität des Saarlandes

von

Joachim Niehren

Saarbrücken
Dezember, 1994

Zusammenfassung

Wir präsentieren den δ-Kalkül, ein Modell von uniform nebenläufiger Berechnung. Er integriert strikte und nicht-strikte funktionale Berechnung und beschreibt in beiden Fällen das gewünschte Komplexitätsverhalten.

Wir nennen nebenläufige Berechnung *uniform nebenläufig*, falls Resultat, Terminierung und Komplexität unabhängig von der Berechnungsreihenfolge sind. Diese Eigenschaften zeigen wir für den δ-Kalkül, indem wir seine uniforme Konfluenz beweisen.

Der δ-Kalkül läßt sich zu Modellen von nebenläufiger Berechnung erweitern, die konsumierbare Resourcen und Indeterminismus zulassen. Dazu gehören der γ-Kalkül, eine Fundierung von nebenläufiger Berechnung mit Constraints, und der π-Kalkül, ein auf Kanalkommunikation basierender Nachfolger von CCS.

Der δ-Kalkül ist ein relationaler Kalkül mit prozeduraler Abstraktion und Applikation. Er ermöglicht Kommunikation über logische Variablen und Suspension auf deren Instantiierung. Beide Mechanismen ergeben sich in natürlicher Art aus paralleler Komposition und Deklaration.

Wir betten den strikten und den nicht-strikten λ-Kalkül in den δ-Kalkül ein. Durch explizite Referenzen stellen wir sicher, daß funktionale Argumente höchstens einmal ausgewertet werden. Explizite Referenzen realisieren wir durch logische Variablen, die wir außerdem zur Darstellung von nicht-strikter funktionaler Kontrolle benötigen. Für die Einbettung des strikten λ-Kalküls beweisen wir Adäquatheit bezüglich Terminierung und Komplexität. Wir vermuten, daß die Einbettung des nicht-strikten λ-Kalküls Terminierung erhält und Komplexität verbessert.

Danksagung

An aller erster Stelle bin ich meinem Doktorvater Gert Smolka zutiefst verbunden. Er hat mich in spannende Forschungsgebiete eingeführt, war an allen Teilen dieser Arbeit aktiv beteiligt und hat mir stets freundschaftlich mit Rat und Tat zur Seite gestanden.

Meinen Freunden und Kollegen Martin Müller und Andreas Podelski verdanke ich viele schöne Stunden mit und ohne Wissenschaft. Mit Andreas habe ich tolle Tage in Paris erlebt. Die Erinnerungen an das Square Trousseau möchte ich nicht missen. Martin und ich haben etliche abwechslungreiche Stunden zum Beispiel bei Viez, Orchideen oder schwitzender Tomatensuppe verbracht. Von seiner Offenheit in Forschungsfragen habe ich sehr profitiert.

Meinen Freunden und Kollegen Konstantin Popov, Wolfgang Bartsch und Tobias Müller bin ich zutiefst zu Dank verpflichtet. Auf ihre Unterstützung konnte ich mich stets verlassen. Kostjas Mitternachtsespresso und das gemeinsame Entschlüsseln kyrillischer Schriftzeichen war mir immer ein großes Vergnügen.

Bei allen Mitarbeitern des Forschungsbereichs Programmiersysteme am Deutschen Forschungszentrum für Künstliche Intelligenz in Saarbrücken möchte ich mich für eine angenehme und engagierte Arbeitsatmosphäre bedanken. An dieser Stelle möchte Christian Schulte, Ralf Treinen, Michael Mehl, Ralf Scheidhauer, Martin Henz und Jörg Würtz nennen.

Meinem Weggefährten Rolf Backofen gebührt Dank für seine Kommentare bezüglich der allerersten Version dieser Arbeit. Aber auch Gert Smolka, Martin Müller, Peter Van Roy, Martin Henz, Kai Ibach, Benjamin Lorenz und Wolfgang Bartsch haben frühere Versionen ganz oder teilweise gelesen und durch ihre Anmerkungen zur Konsistenz beigetragen.

Besonderer Dank gebührt meinen Eltern, die mich auf meinem Weg begleitet haben und stets um mein Wohlergehen besorgt waren. Auch meine Kollegen waren bei Ihnen stets willkommen und konnten dadurch am saarländischen Lebensgefühl teilhaben.

Auch Gerd Wittstock, der Betreuer meiner Diplomarbeit, hat Anteil am Gelingen dieser Dissertation, denn er hat in mir die Freude an wissenschaftlicher Arbeit geweckt. Desweiteren resultiert die Idee zum Schreiben der nachfolgenden Einleitung aus einer gemeinsamen Diskussion.

Finanziell wurde ich während der Erstellung dieser Arbeit durch das Projekt Hydra am Forschungsbereich Programmiersysteme des Deutschen Forschungszentrums für Künstliche Intelligenz (DFKI) und ein Stipendium des Graduiertenkollegs Informatik an der Universität des Saarlandes unterstützt.

Inhaltsverzeichnis

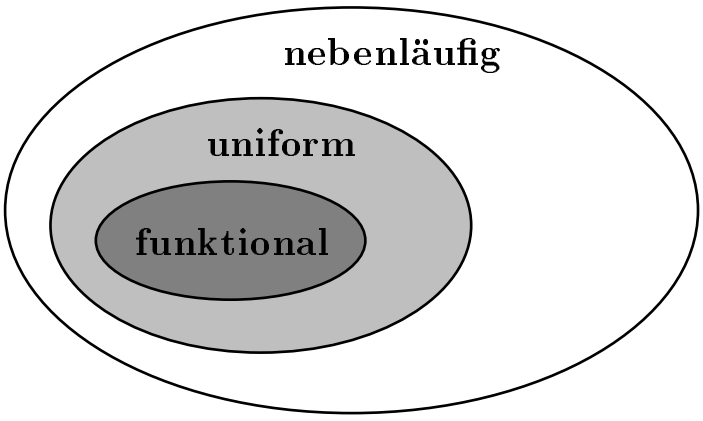
Kapitel 1

Einleitung

Modelle nebenläufiger Berechnung beschreiben das Kommunikations- und Synchronisationsverhalten von nebenläufig rechnenden Prozessen. Das Ziel dieser Arbeit ist, die folgende These zu untermauern:

> Funktionale Berechnung ist eine spezielle Form von nebenläufiger Berechnung, die sich durch Konfluenz und Uniformität auszeichnet, also dadurch, daß Resultat, Terminierung und Komplexität unabhängig von der Berechnungsreihenfolge sind.

Zu diesem Zweck führen wir den δ-Kalkül als Modell von uniformer, konfluenter und nebenläufiger Berechnung ein, die wir kürzer auch *uniform nebenläufige Berechnung* nennen. Desweiteren bilden wir sowohl strikte, als auch nicht-strikte funktionale Berechnung im δ-Kalkül ab, so daß in beiden Fällen Terminierung und Komplexität geeignet reflektiert werden. Zur Modellierung von uniform nebenläufiger Berechnung kombinieren wir Ideen aus zwei Gebieten. Die erste Wurzel dieser Arbeit ist



der π-Kalkül [Mil91, MPW92, EN86], einem Nachfolger von CCS ("Calculus of Communicating Systems") [Mil80], welcher aus der Welt der funktionalen Berechnung hervorgegangen ist. Die zweite Wurzel ist die nebenläufige Programmierung mit Constraints [SR90], insbesondere die Programmiersprache Oz [Smo94d] und der γ-Kalkül [Smo94c].

Der π-Kalkül ist ein kanonisches Modell von nebenläufiger Berechnung, welches Kanalkommunikation als universellen Berechnungsmechanismus etabliert. Kanalkommunikation ermöglicht den Austausch von Ressourcen, wobei mehrere Prozesse um die Ressourcen eines anderen konkurrieren können. Dadurch kann jede Form von Indeterminismus verursacht werden, insbesondere Nicht-Konfluenz und Nicht-Uniformität. Dennoch zeigt Robin Milner im Kontext des π-Kalküls [Mil92], daß sich funktionale Berechnung als spezielle Form von nebenläufiger Berechnung auffassen läßt. Die von ihm vorgestellten Einbettungen des strikten und nicht-strikten λ-Kalküls in den π-Kalkül verwenden eine implizite Synchronisationsdisziplin, welche die Uniformität und Konfluenz der eingebetteten Ausdrücke sicherstellt.

Oz ist eine nebenläufige Sprache mit Constraints, die funktionale und objektorientierte Programmierung [HM94] unterstützt, aber auch enkapsulierte Suche [SS94, MPSW94] und Constraintpropagierung zur Verfügung stellt. Die Konzepte von Oz werden weitgehend durch den Oz-Kalkül [Smo94a, Smo94b] beschrieben. Dieser wurde bis Ende 1992 unter Leitung von Gert Smolka entwickelt und war der Ausgangspunkt dieser Arbeit.

Insbesondere integriert der Oz-Kalkül prozedurale Abstraktion und Applikation mit Constraints erster Ordnung und Kommunikation vergleichbar zum π-Kalkül. Constraints sind hier nur deshalb von Interesse, da sie logische Variablen zur Verfügung stellen. Denn logische Variablen eignen sich hervorragend zur Kommunikation zwischen nebenläufigen Prozessen. Der γ-Kalkül [Smo94c] ist eine Vereinfachung des Oz-Kalküls, der diesen Aspekt von logischen Variablen explizit macht und sich zur Fundierung von funktionaler und nebenläufiger objekt-orientierter Berechnung eignet. Der hier vorgestellte δ-Kalkül ist im wesentlichen eine Einschränkung des γ-Kalküls auf uniform nebenläufige Berechnung.

Der δ-Kalkül modelliert das Komplexitätsverhalten von strikter und nicht-strikter funktionaler Berechnung. Dadurch ist er insbesondere dem nicht-strikten λ-Kalkül überlegen, der mehrfach benötigte funktionale Argumente mehrfach auswertet und sich nur bezüglich Terminierung korrekt verhält. Daß funktionale Argumente im δ-Kalkül nicht mehrfach ausgewertet werden, liegt an der Verwendung von logischen Variablen als Referenzen auf funktionale Argumente. Diese Tatsache ist nicht neu, denn schon Wadsworth [Wad71] stellte 1971 heraus, daß sich das Verdoppeln von funktionalen Argumenten vermeiden läßt, indem man Referenzen auf Argumente übergibt, anstelle der Argumente selbst. Dennoch ist nach Kenntnis des Autors der δ-Kalkül beziehungsweise der γ-Kalkül das erste einheitliche Komplexitätsmodell

von nicht-strikter und strikter funktionaler Berechnung.

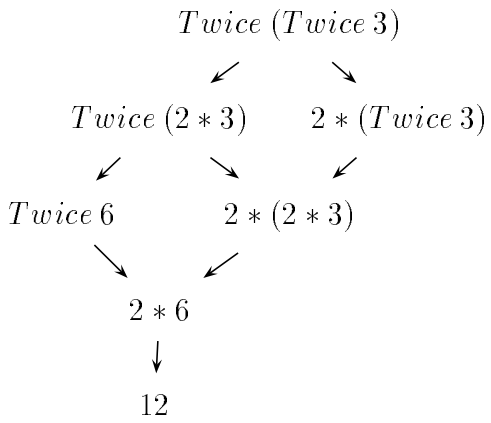
Unsere Ergebnisse für den δ-Kalkül lassen sich so interpretieren, daß Kommunikation über logische Variablen als Kommunikationsmechanismus für uniform nebenläufige Berechnung hinreichend ist, wenn prozedurale Abstraktion und Applikation zur Verfügung stehen. Insbesondere sind Kanäle zu diesem Zweck nicht notwendig, so daß sich der durch diese verursachte Indeterminismus für eine große Klasse von Programmen global ausschließen läßt.

1.1 Problem und Lösungsansatz

Die Idee von funktionaler Berechnung ist, Beschreibungen von Funktionen als Programme aufzufassen. Ein Beispiel dafür ist die arithmetische Funktion $Twice$, die eine ganze Zahl x auf $2 * x$ abbildet, was sich durch folgende Funktionsgleichung beschreiben läßt:

$$Twice\, x \;=\; 2 * x\,.$$

Wir können diese Gleichung aber auch als Beschreibung einer funktionalen Prozedur interpretieren, indem wir sie von links nach rechts orientieren. Rufen wir diese Prozedur mit dem Argument x auf, so gibt sie $2 * x$ zurück. Interessanter ist die zweifache Anwendung auf 3, denn diese führt nicht nur zu einer, sondern zu drei möglichen Berechnungen, die durch folgendes Bild veranschaulicht werden:



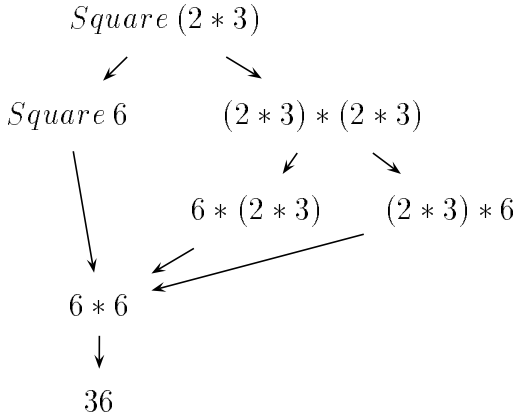
Alle drei Berechnungen terminieren, liefern das gleiche Resultat, nämlich die ganze Zahl 12. Sie haben auch die gleiche Länge, welche uns als Komplexitätsmaß dient. Auch im allgemeinen sind Terminierung, Komplexität und Resultat von funktionalen Programmen unabhängig von der Berechnungsreihenfolge.

Aber leider führen nicht alle Funktionsbeschreibungen der obigen Art zu uniform nebenläufigen Prozeduren, wie wir nun anhand von zwei weiteren Beispielen illu-

strieren werden. Zuerst betrachten wir die Funktion $Square$, die durch

$$Square\, x \;=\; x * x$$

beschrieben wird. Wenden wir $Square$ als funktionale Prozedur auf $2 * 3$ an, dann erhalten wir wiederum drei mögliche Berechnungen:

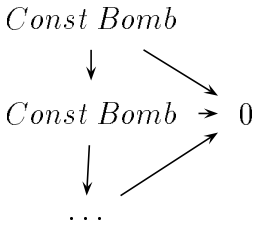


In diesem Fall hängt die Komplexität von der Berechnungsreihenfolge ab, denn in der mittleren und der rechts stehenden Berechnung wird das funktionale Argument $2*3$ zweimal ausgewertet, aber in der links stehenden nur einmal. Wir sagen, daß die Berechnung von $2 * 3$ in der links stehenden Berechnung geteilt wird ("Sharing"), und zwar zwischen allen Auftreten von x in der Beschreibung der Ausgabe von $Square$.

Ein weiteres Problem läßt sich am Beispiel der konstanten Funktion $Const$ beobachten, die durch

$$Const\, x \;=\; 0$$

beschrieben wird. Ist $Bomb$ ein Ausdruck, der durch Reduktion stets auf sich selbst abgebildet wird[1], also $Bomb \rightarrow Bomb$, dann führt die Anwendung der Prozedur $Const$ auf $Bomb$ zu Berechnungen beliebiger Länge:



[1]Im λ-Kalkül können wir $Bomb$ durch $Bomb \;=\; (\lambda x.xx)\,(\lambda x.xx)$ definieren.

In diesem Fall sind weder Terminierung noch Komplexität unabhängig von der Berechnungsreihenfolge, selbst wenn das Ergebnis im Falle der Terminierung eindeutig bestimmt ist.

Der λ-Kalkül und explizite Referenzen. Die meisten abstrakten Modelle funktionaler Berechnung beruhen auf Alonzo Churchs λ-Kalkül [Chu36] und auch alle bisher vorgestellten Beispiele lassen sich durch den λ-Kalkül rechtfertigen. Wir werden in dieser Arbeit zeigen, daß auch alle aufgezeigten Pathologien Probleme des λ-Kalküls sind. Sie hängen alle von der Tatsache ab, daß funktionale Prozeduren im λ-Kalkül *anonym* sind und lediglich durch ihre syntaktische Position referiert werden können.

Insbesondere hat das Fehlen von expliziten Referenzen im λ-Kalkül die folgenden Konsequenzen, die wir später genauer erklären werden:

- Bedarfsgesteuerte Berechnung läßt sich in λ-Kalkülen *nicht* mit dem geeigneten Komplexitätsverhalten modellieren. So werden zum Beispiel im nicht-strikten λ-Kalkül funktionale Argumente *mehrfach* ausgewertet.

- Das Verhalten von Implementierungen nicht-strikter funktionaler Sprachen läßt sich *nicht* in deren Quellsyntax beschreiben. Dadurch lassen sich Auswertungsstrategien verschiedener Implementierungen nur schwer miteinander vergleichen.

- Die Substitution von Ausdrücken für Variablen durch β-Reduktion ist für Implementierungen funktionaler Sprachen *nicht* realistisch; besser ist die Übergabe von Referenzen auf funktionale Argumente, anstelle der Argumente selbst.

- Der strikte λ-Kalkül modelliert zwar Komplexität und Terminierung von strikter funktionaler Berechnung, schränkt aber den Freiheitsgrad der Berechnungsreihenfolge *unnötig* ein.

- Der schwache λ-Kalkül ist *nicht* konfluent.

- Rekursive Prozeduren lassen sich im λ-Kalkül *nicht* direkt ausdrücken.

- Die Integration von logischen Variablen in funktionale Berechnung ist *schwierig*, genauso wie formale Vergleiche mit nebenläufiger und logischer Berechnung.

Der δ-Kalkül. Unabhängig voneinander sind diese Probleme bereits gelöst worden. Der δ-Kalkül bietet jedoch eine einheitliche, abstrakte Lösung an. Diese beruht darauf, alle Ausdrücke des λ-Kalküls mit expliziten Referenzen zu versehen, Funktionen als Relationen mit einem zusätzlichen Ausgabeargument aufzufassen und funktionale Schachtelung durch parallele Komposition und Deklaration von lokalen Variablen aufzulösen.

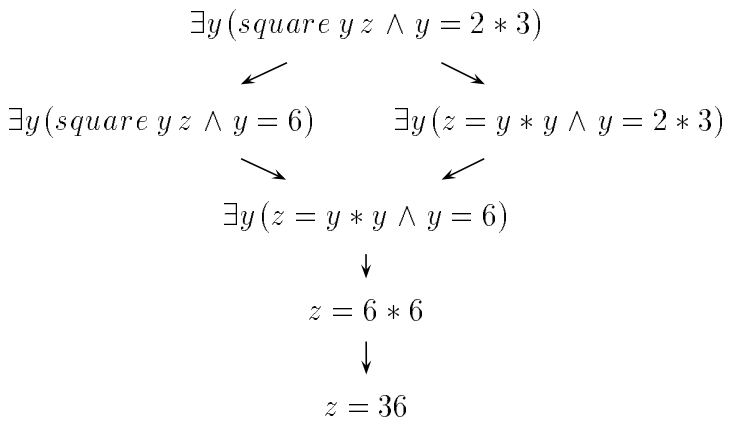
Im δ-Kalkül hat die Definition der Funktion $Square$ des λ-Kalküls folgendes Aussehen, wobei out als formales Ausgabeargument fungiert:

$$square{:}x\,out/out = x * x$$

Die Anwendung von $Square$ auf $2 * 3$ mit Resultat z, kurz $z = Square\ (2 * 3)$, entspricht im δ-Kalkül dem Ausdruck

$$\exists y(square\ y\ z \wedge y = 2 * 3)\ .$$

Alle Berechnungen dieses Ausdruckes haben die gleiche Länge, wobei $2*3$ stets nur einmal ausgewertet wird. Im Gegensatz zum λ-Kalkül ist der Graph aller möglichen Berechnungen im δ-Kalkül vollkommen gleichmäßig.



Auch die Pathologie im Kontext der konstanten Funktion $Const$ löst sich durch explizite Referenzen auf. Das unmittelbare Pendant zu $Const$ im δ-Kalkül ist

$$const{:}x\,out/out = 0\ .$$

Bezüglich dieser Definition führt der Aufruf $z = Const\ Bomb$ zu einer eindeutig bestimmten, unendlichen Berechnung, so daß insbesondere Terminierung und Komplexität unabhängig von der Berechnungsreihenfolge sind.

$$\exists b(const\, b\, z \wedge b = Bomb)$$
$$\downarrow$$
$$z = 0 \wedge \exists b(b = Bomb)$$
$$\downarrow$$
$$z = 0 \wedge \exists b(b = Bomb)$$
$$\downarrow$$
$$\ldots$$

Diese Lösung scheint im erstem Moment nicht zufriedenstellend, da im Vergleich zum λ-Kalkül gerade die endlichen Berechnungen verloren gehen. Der Grund dafür ist, daß *const* sein Argument stets auswertet, ob es gebraucht wird oder nicht.

Wir werden später zeigen, wie wir im δ-Kalkül bedarfsgesteuerte Berechnung darstellen können. Dazu werden wir eine funktionale Prozedur so beschreiben, daß diese die Berechnung ihrer aktuellen Argumente nur bei Bedarf anfordert. Eine Anforderung instantiiert die mit dem Argument assoziierte logische Variable. Diese Darstellung stellt sicher, daß nicht-angeforderte funktionale Argumente auch nicht berechnet werden. Mehrfach angeforderte Argumente werden genau einmal berechnet.

Uniforme Konfluenz. Ein nicht unerheblicher Teil dieser Arbeit besteht aus der Untersuchung von abstrakten Berechnungskalkülen. Darunter verstehen wir Ableitungssysteme, bestehend aus einer Menge von Ausdrücken und einer Reduktionsrelation $\rightarrow$ auf Ausdrücken[2].

Eine Ableitung in einem Kalkül ist eine endliche oder unendliche Folge von Ausdrücken, die durch Reduktion auseinander hervorgehen. Unter einer Berechnung verstehen wir eine maximale endliche Ableitung oder eine unendliche Ableitung. Eine Berechnung eines Ausdruckes N ist eine Berechnung, die mit N beginnt. Wir nennen einen Ausdruck uniform, wenn all seine Berechnungen die gleiche Länge haben. Ein Kalkül heißt uniform, wenn all seine Ausdrücke uniform sind. Ein Kalkül heißt konfluent, wenn je zwei endliche Ableitungen desgleichen Ausdruckes stets zusammengeführt werden können.

Ein zentrales Kriterium sowohl für Uniformität als auch für Konfluenz ist uniforme Konfluenz. Wir nennen einen Kalkül *uniform konfluent*, falls er die Eigenschaft in Abbildung 1.1 erfüllt: Für beliebige Ausdrücke M, N_1 und N_2 mit $N_1 \leftarrow M \rightarrow N_2$ gilt $N_1 = N_2$ oder es existiert ein Ausdruck N mit $N_1 \rightarrow N \leftarrow N_2$. So sind zum Beispiel der strikte und nicht-strikte λ-Kalkül uniform konfluent. Der Begriff der

[2]Später werden wir eine Äquivalenzrelation $\equiv$ auf Ausdrücken hinzunehmen, um von der Gleichheit auf der Menge der Ausdrücke zu abstrahieren.

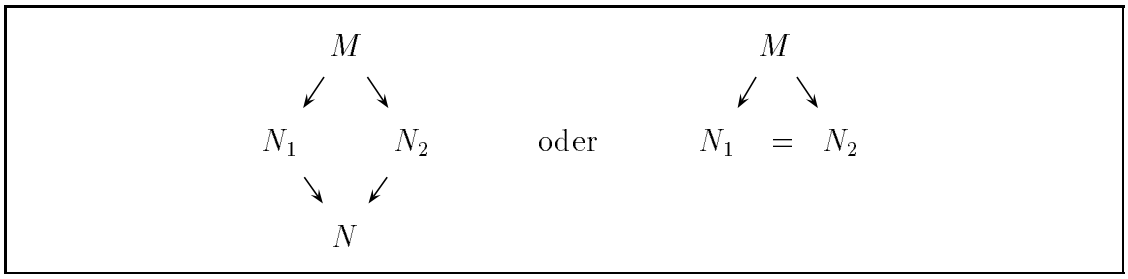


Abbildung 1.1: Uniforme Konfluenz

uniformen Konfluenz ist unser zentrales Hilfsmittel beim Beweis der Adäquatheit von Kalküleinbettungen.

1.2 Funktionale Berechnung

Die meisten Modelle funktionaler Berechnung sind Varianten von Churchs λ-Kalkül [Chu36], der gleichzeitig der Ursprung der funktionalen Idee ist. Die Ausdrücke M, N des λ-Kalküls sind Variablen, Abstraktionen oder Applikationen:

$$M, N \ ::= \ x \mid \lambda x.M \mid MN$$

Abstraktionen beschreiben funktionale Prozeduren; so ist zum Beispiel $Twice = \lambda x.2 * x$ gleichbedeutend zur Beschreibung von $Twice$ ganz zu Beginn. Hier machen wir jedoch explizit, daß $Twice$ und $=$ lediglich Konstrukte der Metasyntax sind, und nicht zur Syntax des λ-Kalküls gehören. Weiterhin verwenden wir in λ-Ausdrücken zusätzliche Syntax für vordefinierte Datenstrukturen ohne deren Semantik zu erklären. Dies ist zwar nicht wesentlich für die betrachteten Phänomene, ermöglicht aber intuitivere Beispiele. Insofern machen wir implizit Gebrauch von der wohlbekannten Tatsache, daß sich beliebige Datenstrukturen durch Abstraktionen darstellen lassen[3]. Es ist eine wesentliche Idee des λ-Kalküls, daß sich die gleiche Sprache zur Beschreibung von Prozeduren und Daten aus scheinbar verschiedenen semantischen Bereichen verwenden läßt.

Die Reduktion des λ-Kalküls und all seiner Varianten basiert auf dem (β)-Axiom:

$$(\lambda x.M)N \ \rightarrow_\beta \ M[N/x]$$

Dabei steht $M[N/x]$ für denjenigen Ausdruck, den man erhält, wenn man alle Auftreten von x in M durch N ersetzt[4].

[3]Die Korrektheit der üblichen Kodierungen gilt meist nur bezüglich β-Gleichheit (z.B in [Bar90]) und nicht bezüglich Terminierung oder Komplexität.

[4]Wir vernachlässigen hier Inkonsistenzen, die bei Substitution durch Variablenbinder ent-

Funktionale Programme verwenden intensiv Prozeduren höherer Ordnung. Darunter verstehen wir solche Prozeduren, deren Argumente und Resultate selbst wieder Prozeduren sein können. So beschreibt zum Beispiel

$$ApplyTwice \;=\; \lambda f.\lambda x.\; f\;(f\;x)$$

eine Prozedur, die als Argument eine Prozedur f erwartet und dann $\lambda x.f\;(f\;x)$ ausgibt. Diese Ausgabe ist wiederum eine Prozedur, die f zweimal auf ihr Argument x anwendet. Bezeichnen wir mit I die λ-Identität $\lambda x.x$, dann erhalten wir die folgende Berechnung:

$$(ApplyTwice\;I)\;I \;\;\rightarrow_\beta\;\; (\lambda x.I\;(I\;x))\;I \;\;\rightarrow_\beta\;\; I\;(II) \;\;\rightarrow_\beta\;\; II \;\;\rightarrow_\beta\;\; I$$

Berechnung mit Prozeduren höherer Ordnung kann also dynamisch neue Prozeduren erzeugen, wie $\lambda x.I\;(I\;x)$ im voranstehenden Beispiel.

Der Substitutionsoperator $[N/x]$. Der komplexe Substitutionsoperator $[N/x]$ in (β) ist problematisch im Gegensatz zu $[y/x]$. Für Implementierungen funktionaler Sprachen ist der komplexe Operator nicht realistisch und in Modellen von funktionaler Berechnung verursacht er Komplexitätsprobleme.

Die Komplexitätprobleme treten auf, da Anwendungen von $[N/x]$ den eventuell noch nicht berechneten Ausdruck N kopieren können. Dies führt zur wiederholten Auswertung von N, wie zum Beispiel von $2*3$ in $Square\;(2*3) \rightarrow (x*x)[2*3/x]$. Andererseits läßt sich der komplexe Substitutionsoperator im λ-Kalkül mangels expliziter Referenzen nicht vermeiden.

Auch für Implementierungen funktionaler Sprachen wäre Kopieren von beliebigen Ausdrücken eine komplexe Operation. Denn würde zum Beispiel $(xx)[N/x]$ wörtlich ausgeführt, dann müßte diejenige interne Struktur kopieren werden, die N darstellt. Kopieren ist jedoch üblicherweise nur für solche Strukturen vorgesehen, die Abstraktionen darstellen. Dabei denken wir an das Abspulen eines Prozedurrumpfes bei Applikation. Für allgemeinere Strukturen ist Kopieren aufwendig. Dennoch werden solche Kopiermechanismen in anderen Kontexten realisiert. Dazu gehören Implementierungen von nebenläufigen Suchmechanismen [SS94, SSW94, MPSW94, JH91, Jan94] und von kopierender Speicherbereinigung.

Der schwache λ-Kalkül. Im reinen λ-Kalkül darf (β) in beliebigen Teilausdrücken angewendet werden, was zu einem konfluenten System führt. Zur Modellierung des Laufzeitverhaltens von funktionaler Berechnung ist Reduktion innerhalb von Abstraktionen jedoch nicht sinnvoll. Denn im reinen λ-Kalkül lassen sich

stehen können. Eine Möglichkeit eine formal korrekte Definition zu erhalten, ist anstelle von λ-Ausdrücken Äquivalenzklassen von λ-Ausdrücken modulo α-Konversion zu betrachten und β-Reduktion auf α-standardisierten Repräsentanten zu definieren.

rekursive Prozeduren nicht so formulieren, daß alle erlaubten Berechnungen terminieren.

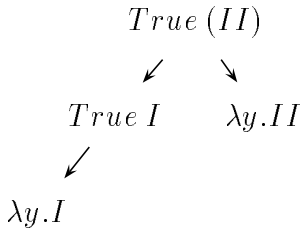
Betrachten wir zum Beispiel die Fakultätsfunktion, die sich in einer funktionalen Sprache mit Konditionalen wie folgt beschreiben läßt:

$$Fak\, n \;=\; \mathsf{if}\ n > 0\ \mathsf{then}\ n * Fak\,(n-1)\ \mathsf{else}\ 1\ \mathsf{fi}$$

Würden wir dieses Konditional als dreistellige Funktion im reinen λ-Kalkül kodieren, so dürften wir in den Zweigen des Konditionals frei reduzieren. Dann könnten wir zum Beispiel die folgende unendliche Berechnung nicht verhindern:

$$\begin{array}{rcl} Fak\,0 & \rightarrow & \mathsf{if}\ 0 > 0\ \mathsf{then}\ 0 * Fak\,(-1)\ \mathsf{else}\ 1\ \mathsf{fi} \\ & \rightarrow & \mathsf{if}\ 0 > 0\ \mathsf{then}\ 0 * (\mathsf{if}\ (-1) > 0\ \mathsf{then}\ (-1) * Fak\,(-2)\ \mathsf{else}\ 1\ \mathsf{fi})\ \mathsf{else}\ 1\ \mathsf{fi} \\ & \rightarrow & \ldots \end{array}$$

Aus diesem Grund lassen wir, wie in verwandten Arbeiten üblich, Reduktion in Abstraktionen nicht zu. Die resultierende Variante des reinen λ-Kalküls heißt schwacher λ-Kalkül. Dieser ist unglücklicherweise nicht konfluent, wie schon die beiden verschiedenen Berechnungen von $True\,(II)$ mit $True = \lambda x.\lambda y.\, x$ zeigen:



Der unmittelbare Grund für diese Anomalie ist, daß β-Reduktion einen reduzierbaren Ausdruck in eine Position bewegen kann, in der Reduktion verboten ist.

1.2.1 Der strikte λ-Kalkül

Das Konfluenzproblem des schwachen λ-Kalküls läßt sich im Rahmen des λ-Kalküls lösen indem man β-Reduktion weiter einschränkt. Der strikte λ-Kalkül ("eager" oder "call-by-value") ist diejenige Variante des schwachen λ-Kalküls, in der (β) nur dann angewendet werden darf, falls das Argument eine Abstraktion ist. Wie wir bereits angemerkt haben, ist der strikte λ-Kalkül uniform konfluent.

Der strikte λ-Kalkül erzwingt, daß funktionale Argumente genau einmal berechnet werden, unabhängig davon, ob sie mehrfach oder überhaupt nicht benötigt werden. Alle Anforderungen eines Arguments teilen sich somit seine Berechnung. Im strikten λ-Kalkül läßt sich also "Sharing" von Berechnung erzwingen. Dies ist eine

wesentliche Eigenschaft von Komplexitätsmodellen. In der Tat ist der strikte λ-Kalkül ein geeignetes Komplexitätsmodell für strikte funktionale Berechnung wie sie zum Beispiel von ML [HMM86], Lisp [WH81] und Scheme [CR91] angeboten wird.

In den bisher vorgestellten Beispielen für Berechnung im λ-Kalkül, $Twice$, $Square$, $Const$ und $True$ ist jeweils genau die am weitesten links stehende Berechnung strikt. Im allgemeinen ist jedoch die Berechnungsreihenfolge des strikten λ-Kalküls nicht deterministisch. Ist zum Beispiel I die λ-Identität $\lambda x.x$, dann läßt der Ausdruck $(II)\,(II)$ zwei verschiedene Reduktionsschritte zu.

1.2.2 Der nicht-strikte λ-Kalkül

Im Kontext von Berechnung ist man häufig an unendlichen Objekten interessiert, die sich endlich darstellen lassen. Beispiele dafür sind die Liste aller natürlichen Zahlen und die Menge der von einem endlichen Automaten erzeugten Wörter. Kennt man eine endliche Beschreibung eines unendlichen Objektes, so benötigt man einen terminierenden Algorithmus, der die Berechnung von Teilen des Objektes bei Bedarf ermöglicht. Auch für große endliche Objekte, die sich kompakt beschreiben lassen, ist bedarfsgesteuerte Berechnung von Objektteilen nützlich.

Die Idee von nicht-strikten funktionalen Sprachen ist es, Berechnung von funktionalen Argumenten ausschließlich bei Bedarf auszuführen, so daß deren Anforderung implizit bleibt[5]. Dies steht im Gegensatz zu strikten funktionalen Sprachen, in denen funktionale Argumente stets ausgewertet werden, ob sie gebraucht werden oder nicht.

Betrachten wir zum Beispiel die Prozedur $NatList$, die die Liste aller natürlichen Zahlen beschreibt, die größer als n sind.

$$NatList\, n \;=\; n|(NatList\, n+1)$$

Die Auswertung von $NatList\, 0$ in einer strikten Sprache würde nicht terminieren. In einer nicht-strikten Sprache hingegen würde sie terminieren und zwar mit $0|(NatList\, 0+1)$. Nur durch Selektion der zweiten Listenkomponente könnte die Auswertung von $NatList\, 0+1$ angestoßen werden.

[5]Die Begriffe "strikte" und "nicht-strikte funktionale Berechnung" sind semantisch motiviert, aber aus operationaler Sicht nicht wirklich befriedigend. Denn wesentlich für nicht-strikte funktionale Berechnung ist, daß potentielle Berechnungen nur bei Bedarf angefordert werden, und nicht etwa, daß unendliche Berechnungen weggeworfen werden, falls ihr Resultat nicht benötigt wird. Die englischen Begriffe "eager" und "lazy" treffen diesen Sachverhalt besser als die Begriffe "strikt" und "nicht-strikt", lassen sich aber nicht vernünftig ins Deutsche übersetzen.

Der nicht-strikte λ-Kalkül ("lazy λ-calculus" oder "call-by-name λ-calculus")[6] ist diejenige Variante des schwachen λ-Kalküls, die (β)-Reduktion in Argumentposition verbietet. Dadurch werden funktionale Argumente nur dann ausgewertet, wenn sie benötigt werden, in diesem Fall aber auch genauso oft, wie sie benötigt werden.

So ist zum Beispiel die kürzeste terminierende Berechnung von $ConstBomb$ nicht-strikt, genauso, wie die Berechnung $True\,(II) \to \lambda y.II$. Das einfachste Beispiel für die wiederholte Auswertung eines Argumentes ist:

$$(\lambda x.xx)\,(II) \;\to\; (II)\,(II) \;\to\; I\,(II) \;\to\; II \;\to\; I\,.$$

Das gleiche Phänomen tritt in der mittleren und in der rechts stehenden Berechnung von $Square\,(2*3)$ auf; allerdings sind diese beiden Berechnungen nicht ganzheitlich nicht-strikt, da die vordefinierte Multiplikation nur strikt Sinn macht.

Im nicht-strikten λ-Kalkül läßt jeder Ausdruck genau eine Berechnung zu, was natürlich uniforme Konfluenz sicherstellt. In der Tat liefert der nicht-strikte λ-Kalkül ein Terminierungsmodell für nicht-strikte funktionale Sprachen, wie zum Beispiel Haskell [HWA+90], Miranda [Tur85], Id [Nik91] und Concurrent Clean [EHN+93].

Als Komplexitätsmodell von nicht-strikter funktionaler Berechnung ist der nicht-strikte λ-Kalkül natürlich *nicht* geeignet, da er funktionale Argumente unter Umständen mehrfach auswertet. Dieses Problem ist schon lange bekannt, aber erst vor kurzem auf hoher Abstraktionsebene gelöst worden. Für eine weitergehende Diskussion dieses Themas verweisen wir auf Abschnitt 1.4.1 dieser Einleitung.

Interessanterweise läßt sich nicht-strikte Berechnung auch im strikten λ-Kalkül darstellen. Auch dies ist schon seit langem bekannt, denn der nicht-strikte λ-Kalkül läßt sich in den strikten einbetten [DH92, Plo75]. Dabei bleiben allerdings die Komplexitätsprobleme des nicht-strikten λ-Kalküls erhalten.

---

[6] Die Begriffe "lazy" und "call-by-name" werden häufig auch anders verwendet. Mit "lazy" verbindet man auch bedarfsgesteuerte Berechnung, die nicht mit den Komplexitätsproblemen des "lazy λ-calculus" behaftet ist. In [AFM+95] wird ein in diesem Sinne geeigneter λ-Kalkül zur klaren Unterscheidung mit "call-by-need λ-calculus" bezeichnet.
Der Begriff "call-by-name" beschreibt im ursprünglichen Sinne die Parameterübergabe im Stile von Algol68. Manchmal wird er auch mit "dynamic Binding" in Beziehung gesetzt, also mit der textuellen Ersetzung von aktuellen für formale Argumente, ohne Berücksichtigung von Variablenbindern. Die Bezeichnung "lazy λ-calculus" ist dadurch motiviert, daß in diesem Kalkül aktuelle Argumente für formale substituiert werden, jedoch nach vorheriger konsistenter Variablenumbenennung ("static Binding").
Die Bezeichnung "call-by-name λ-calculus" ist inbesondere deshalb irreführend, weil Funktionen in λ-Kalkülen ohne *let*-Ausdrücke keine Namen besitzen. Das Fehlen von Namen im Kalkül selbst verursacht paradoxerweise gerade die Komplexitätsproblematik des "call-by-name λ-calculus".

1.3 Uniform nebenläufige Berechnung

Der δ-Kalkül verwendet relationale Schreibweisen zur Modellierung von uniform nebenläufiger Berechnung, die insbesondere strikte und nicht-strikte funktionale Berechnung umfaßt. Relationale Schreibweisen werden auch in vielen anderen formalen Sprachen verwendet. Dazu zählen Grammatiken, logische Programmiersprachen, Beweiskalküle für Logik erster Stufe und andere Modelle nebenläufiger Berechnung.

Insofern schlägt der δ-Kalkül eine Brücke zwischen formal verschiedenenartigen Systemen. Dabei vermeidet er Indeterminismus wie zum Beispiel durch "committed Choice" und Nichtdeterminismus wegen disjunktiver Information.

1.3.1 Der δ-Kalkül

Ausdrücke E, F des δ-Kalküls sind benannte Abstraktionen, Applikationen, Gleichungen zwischen Variablen, (parallele) Kompositionen und Deklarationen; diese definieren wir durch

$$E, F \ ::= \ x{:}\overline{y}/E \mid x\,\overline{y} \mid x = y \mid E \wedge F \mid \exists x\, E\,,$$

wobei $\overline{y}$ für eine endliche Folge von Variablen steht.[7] Wir identifizieren alle Ausdrücke, die sich bis auf übliche Eigenschaften von Deklaration, Komposition und Variablenbindern nicht unterscheiden. Solche Ausdrücke nennen wir kongruent und schreiben $E \equiv F$.

Reduktion ist im δ-Kalkül in allen Positionen erlaubt, aber nicht unterhalb von Abstraktion. Im Vergleich zum λ-Kalkül ersetzen wir (β) durch Applikation (A):

$$x{:}\overline{y}/E \wedge x\,\overline{z} \ \rightarrow_A \ x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}]$$

Der wesentliche Unterschied zu (β) besteht darin, daß (A) Variablen für Variablen substituiert und *nicht* Ausdrücke für Variablen. Es werden also Referenzen auf Argumente übergeben und nicht die Argumente selbst. Dies stellt sicher, daß bei der Anwendung von Prozeduren keine Berechnung kopiert werden kann.

Der Umgang mit Referenzen wird im δ-Kalkül vollständig durch Elimination (E) beschrieben, welche Ketten von *lokalen* Referenzen verkürzt.

$$\exists x \exists y (x = y \wedge E) \ \rightarrow_E \ \exists y\, E[y/x] \qquad \text{falls } x \text{ und } y \text{ verschieden sind.}$$

[7] Klammern verwenden wir für δ-Ausdrücke optional. Wenn wir diese weglassen, dann hat Abstraktion eine höherer Priorität als Komposition. Zum Beispiel steht $x{:}\overline{y}/E \wedge x\,\overline{z}$ für $(x{:}\overline{y}/E) \wedge x\,\overline{z}$ und nicht für $x{:}\overline{y}/(E \wedge x\,\overline{z}\,)$.

Der Effekt von Elimination ist ähnlich zu demjenigen von Unifikation erster Stufe [LMM88] in Abwesenheit von Termkonstruktoren. Denn beide Mechanismen etablieren durch Gleichungen beschriebene Gleichheit von Variablen. Der wesentliche Unterschied besteht darin, daß die beteiligten Variablen in (E) lokal sind. Dadurch vermeiden wir etliche delikate Schwierigkeiten in Konfluenzbeweisen [NS94], die durch Unifikation mit globalen Variablen verursacht würden.

Um Berechnung im δ-Kalkül formal zu illustrieren, reduzieren wir den Ausdruck $\exists y\,(y = I \wedge x = (yy)\,y)$, wobei wir $y = I$ als Abkürzung für $y{:}u\,v/v = u$ und $x = (yy)\,y$ anstelle von $\exists z\,(y\,y\,z \wedge z\,y\,x)$ schreiben[8].

$$\begin{array}{rcl}
\exists y\,(y = I \wedge \exists z\,(y\,y\,z \wedge z\,y\,x)) & \equiv & \exists y \exists z\,(y = I \wedge y\,y\,z \wedge z\,y\,x) \\
& \to_A & \exists y \exists z\,(y = I \wedge z = y \wedge z\,y\,x) \\
& \equiv & \exists y \exists z\,(z = y \wedge y = I \wedge z\,y\,x) \\
& \to_E & \exists y\,(y = I \wedge y\,y\,x) \\
& \to_A & \exists y\,(y = I \wedge x = y)
\end{array}$$

In der Tat verhält sich diese Berechnung ähnlich zu derjenigen von $(II)\,I$ im λ-Kalkül; denn durch zweimalige Applikation bindet sie x an I. Dieses Beispiel zeigt, wie Applikation, Elimination und Kongruenz interagieren. Applikation baut Referenzketten auf, die durch Elimination verkürzt werden, was wiederum Applikation ermöglicht. Wir weisen darauf hin, daß wir vor jeder Anwendung von (E) und (A) zu einem geeigneten Repräsentanten modulo Kongruenz $\equiv$ übergehen müssen.

Inkonsistenzen. Wir haben bisher verschwiegen, daß nicht alle Ausdrücke E Ausdrücke des δ-Kalküls sind. Denn ohne weitere Einschränkung wäre der bisher vorgestellte Kalkül nicht konfluent, also auch nicht uniform konfluent. Das Problem sind inkonsistente Ausdrücke der Form $x{:}\overline{y}/E \wedge x{:}\overline{z}/F$. Denn in solchen Ausdrücken referiert x auf zwei verschiedende Abstraktionen, so daß eine Applikation von x die freie Auswahl hat, mit welcher sie reduziert.

Um dieses Problem zu umgehen, erlauben wir im δ-Kalkül nur solche Ausdrücke, deren Reduktion niemals zu Inkonsistenzen führt. Diese Eigenschaft ist für beliebige Ausdrücke unentscheidbar. Wir werden jedoch nachweisen, daß die Ausdrücke, die aus Einbettungen von λ-Kalkülen hervorgehen, niemals zu Inkonsistenzen führen.

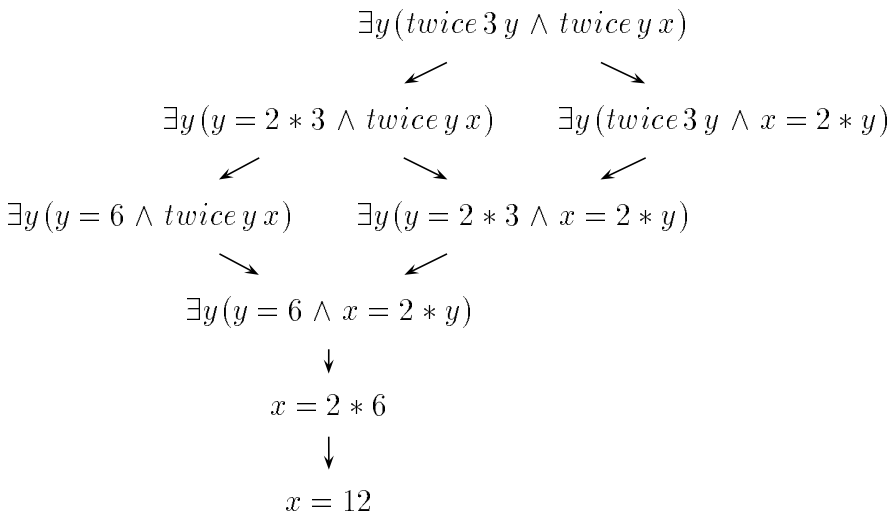
Vergleich mit dem strikten λ-Kalkül. Die bisher vorgestellten Beispiele haben bereits illustriert, wie sich der strikte λ-Kalkül in den δ-Kalkül vermöge Benennung einbetten läßt. Ein wichtiges technisches Resultat dieser Arbeit ist der Nachweis der Adäquatheit dieser Einbettung bezüglich Komplexität und Terminierung. Wir werden zeigen, daß jede Berechnung im strikten λ-Kalkül genauso viele (β)-Schritte benötigt, wie ihr Pendant im δ-Kalkül (A)-Schritte.

[8] In diesem Ausdruck ist x eine Referenz auf $\mathsf{let}\ y{=}I\ \mathsf{in}\ (yy)\,y$.

Dennoch ist Berechnung im δ-Kalkül flexibler als im strikten λ-Kalkül. Um dies zu illustrieren, greifen wir auf unser allererstes Beispiel mit der Funktion $Twice$ zurück. Eine strikte Beschreibung von *twice* im δ-Kalkül - unter Verwendung von vordefinierten Datenstrukturen - ist die Abstraktion

$$twice{:}x\,out/out = 2 * x\ .$$

Der Aufruf $x = Twice\,(Twice\,3)$ entspricht $\exists y(twice\,3\,y \wedge twice\,y\,x)$. Im Stile unserer einführenden Beispiele stellen wir die Berechnungen dieses Ausdruckes graphisch dar:



Die Topologie dieses Graphen stimmt mit der Topologie des Berechnungsgraphen von $Twice(Twice3)$ im reinen λ-Kalkül überein, bis auf den Knoten, der durch die Elimination von y verursacht wird. Jedoch hat nur die am weitesten links stehende Berechnung ein Pendant im strikten λ-Kalkül. Dadurch wird die Adäquatheit der Einbettung des strikten λ-Kalküls jedoch nicht beeinträchtigt, denn alle Berechnungen im δ-Kalkül verhalten sich bezüglich Komplexität und Terminierung gleich.

1.3.2 Nebenläufigkeit und logische Variablen

Unter Nebenläufigkeit verstehen wir, daß verschiedene Berechnungsaktivitäten nebeneinander ausgeführt und beobachtet werden können. Es spielt keine Rolle, ob diese Aktivitäten physikalisch parallel, oder aber im Wechsel vorangetrieben werden.

Berechnungsaktivitäten nennt man üblicherweise Prozesse. Verschiedene Prozesse können miteinander kommunizieren und sich synchronisieren indem sie auf Information von anderen warten. Solange diese Information nicht bekannt ist, kann ein Prozeß die davon betroffenen Unterprozesse blockieren.

Strikte funktionale Berechnung ist nebenläufig. Denn steht zum Beispiel I für die λ-Identität, dann können im Ausdruck $(II)(II)$ Funktion und Argument unabhängig voneinander berechnet werden. Kommunikation findet einzig und allein durch Anwendung von (β) statt, wobei einer Funktion ihr Argument mitgeteilt wird. Ein im Rumpf einer Abstraktion lokalisierter Prozeß ist blockiert und wartet auf eine Anwendung der umgebenden Abstraktion.

Der δ-Kalkül ist ein besseres Modell von nebenläufiger Berechnung als der strikte λ-Kalkül. Er löst nicht nur überflüssige Abhängigkeiten zwischen Funktion und Argument auf, sondern besitzt mächtigere Kommunikations- und Synchronisationsmechanismen. Zur Kommunikation stehen zusätzlich logische Variablen zur Verfügung und zur Synchronisation das Warten auf deren Instantiierung ("Suspension").

Kommunikation über logische Variablen. Im Kontext von Logik erster Stufe versteht man unter einer logischen Variablen einen Stellvertreter für einen Wert des semantischen Bereiches. Dieser Wert ist typischerweise nicht, oder nur unvollständig beschrieben. Im Kontext des δ-Kalküls müssen wir diese Charakterisierung von logischen Variablen leicht variieren, da wir nicht ohne weiteres einen semantischen Bereich angeben können[9].

Im folgenden verstehen wir unter einer logischen Variablen eine Referenz auf eine Abstraktion. Diese Abstraktion muß nicht explizit angegeben sein und kann durch Reduktion berechnet werden. Gemäß dieser Beschreibung verhalten sich alle Variablen des δ-Kalküls wie logische Variablen. Wir sagen, daß eine Variable x in E gebunden ist, falls x in E eine Abstraktion referiert. Zum Beispiel ist x in $x{:}\overline{y}/E$ und in $\exists u(x = u \wedge u{:}\overline{y}/E)$ gebunden.

Im δ-Kalkül entsteht ein Prozeß durch Reduktion eines Ausdruckes. Ein Prozeß kann eine logische Variable an eine Abstraktion binden, so daß andere Prozesse über diese Variable auf diese Abstraktion zugreifen können. Durch diesen Mechanismus kann ein Prozess mit vielen anderen kommunizieren ("one-to-many Communication"). Bindungen derselben Variablen an verschiedene Abstraktionen führen zur Inkonsistenz. Mehrfache Bindungen an dieselbe Abstraktion[10] sind hingegen unkritisch. Dies ist ein wertvolles Ausdrucksmittel, denn es ermöglicht die Kommunika-

[9] Semantische Bereiche für λ-Kalküle werden zum Beispiel in [Win93, GS90, Mos90] konstruiert.

[10] Exakter formuliert meinen wir hier Bindungen an das gleiche Auftreten einer Abstraktion. Dadurch vermeiden wir komplexere Gleichheitsrelationen auf Abstraktionen.

tion von mehreren Prozessen mit einem ("many-to-one Communication"). Ferner kann ein Prozeß darauf warten, daß eine logische Variable gebunden wird ("Suspension"). Kommunikation und Suspension ermöglich die Synchronisation von Prozessen.

Als Beispiel betrachten wir die von den Ausdrücken E, F_1 und F_2 erzeugten Prozesse, wobei wir das Verhalten der Variablen var beobachten wollen:

$$\begin{array}{rcl} E & \equiv & \exists v \exists w\,(var\,v\,w \wedge E') \\ F_1 & \equiv & var = i \\ F_2 & \equiv & var = i \end{array}$$

Wir setzen voraus, daß var in E' ungebunden ist. Somit ist var auch in E ungebunden. Desweiteren ist var in F_1 und F_2 bekannt. Nun betrachten wir eine Berechnung von:

$$G_1 \;\equiv\; \exists i \exists var(i = I \wedge E \wedge F_1 \wedge F_2)\,.$$

Durch Elimination von i wird var sowohl von F_1 als auch von F_2 an eine für E, F_1 und F_2 globale Abstraktion I gebunden, ohne eine Inkonsistenz zu verursachen[11].

$$G_1 \;\rightarrow_E\; \exists var(var = I \wedge E) \;\equiv\; G_2$$

Die Berechnung der Prozesse F_1 und F_2 ist nun schon beendet. Nun reduzieren wir die ehemals suspendierte Applikation $var\,v\,w$ in E, wodurch die Gleichung $w = v$ erzeugt wird:

$$G_2 \;\rightarrow_A\; \exists var \exists v \exists w(var = I \wedge w = v \wedge E')$$

Die Gleichung $w = v$ kann weitere Berechnungen in E' auslösen.

Bedarfsgesteuerte Berechnung. Logische Variablen ermöglichen die Darstellung von Berechnung bei Bedarf mit dem gewünschten Komplexitätsverhalten. Die Idee dazu ist, einen zu berechnenden Ausdruck so darzustellen, daß seine Berechnung nur auf eine explizite Anforderung warten muß und bei Bedarf Anforderungen abzusetzen.

Wollen wir zum Beispiel $y = II$ bei Bedarf berechnen, so definieren wir einen Ausdruck $y \cdot s = II$, der die Berechnung von y startet, sobald s an ein Auftreten der λ-Identität I gebunden wird:

$$y \cdot s = II \;\equiv\; \exists u \exists u' \exists z(u = I \wedge z = I \wedge u'\,z\,y \wedge s\,u\,u')$$

Dabei setzen wir voraus, daß alle auftretenden Variablen paarweise verschieden sind. Dies stellt sicher, daß $y \cdot s = II$ irreduzibel und s global ist. Die folgende

[11] Wir können i eliminieren indem wir Elimination (E) mit der Gleichung $var = i$ anwenden und dann i zu var umbenennen.

Sequenz zeigt, welche Berechnung die Bindung von s an I auslöst:

$$\begin{array}{lll} \exists s\,(s = I \wedge y \cdot s = II) & \rightarrow_A & \exists s \exists u \exists u' \exists z\,(s = I \wedge u = I \wedge z = I \wedge u'\,z\,y \wedge u' = u) \\ & \rightarrow_E & \exists s \exists u \exists z\,(s = I \wedge u = I \wedge z = I \wedge u\,z\,y) \\ & \rightarrow_A & \exists s \exists u \exists z\,(s = I \wedge u = I \wedge z = I \wedge y = z) \\ & \equiv & \exists z\,(y = z \wedge z = I) \wedge \exists s \exists u\,(\ldots) \end{array}$$

Nun ist y in der Tat an das Ergebnis der Berechnung von II gebunden und somit kann auf dieses Ergebnis beliebig oft zugegriffen werden ohne die Berechnung zu wiederholen.

Im allgemeinen stellen wir nicht-strikte funktionale Prozeduren so dar, daß sie die Berechnung von benötigten Argumenten explizit anfordern. Argumente beschreiben wir stets durch verzögerte Ausdrücke ("Thunks" in [DH92]). Um Argumente anstoßen zu können, liest eine nicht-strikte Prozedur mit jedem Argument eine logische Variable ein. Über diese Variablen setzt sie bei Bedarf Anforderungen mittels Instantiierung ab. Wird ein Argument mehrfach angefordert, dann wird die mit ihm assoziierte logische Variable mehrfach gebunden. Wir stellen Konsistenz dadurch sicher, daß wir solche Variablen stets an die gleiche Abstraktion binden.

1.4 Ein Vergleich mit verwandten Arbeiten

1.4.1 Nicht-strikte Komplexitätsmodelle

Als Komplexitätsmodell von nicht-strikter funktionaler Berechnung ist der nicht-strikte λ-Kalkül nicht geeignet. Dieses Problem wurde lange Zeit ausschließlich auf der Ebene von Implementierungstechniken gelöst, wozu Graphreduktion ("Graph Reduction") [PJ87] [PJ87]und Kalküle mit expliziten Substitutionen [Yos93, PS92, ACCL91] gehören. Für eine knappe Einführung in solche Implementierungstechniken verweisen wir auf [Mar90]. Konfluenzuntersuchungen für Kalküle mit expliziten Substitutionen finden sich in [Les94].

Erst vor kurzem ist es John Launchbury [Lau93] gelungen, ein abstraktes Modell für nicht-strikte funktionale Berechnung mit dem gewünschten Komplexitätsverhalten zu finden. Seine Lösung verwendet partielle Abbildungen zur Darstellung von Umgebungen ("Environments" oder "Closures") und ist im Stile einer natürlichen Semantik ("natural Semantics" oder "big-step Semantics") formuliert. Sie stellt sicher, daß ein Ausdruck der Umgebung genau dann ausgewertet wird, wenn sein Resultat benötigt wird. John Launchbury beweist mit bereichstheoretischen ("Domain Theory") Methoden, daß sein Kalkül die gleichen Terminierungseigenschaften besitzt, wie der nicht-strikte λ-Kalkül. Beweise von zusätzlichen Komplexitätaussagen lassen sich mit bereichstheoretischen Methoden jedoch nicht erwarten.

In [AFM+95] wird eine Ein-Schritt-Semantik für nicht-strikte funktionale Berechnung mit "Sharing"vorgeschlagen, der sogenannte "call-by-need λ-calculus". Dieser verwendet *let*-Ausdrücke zur Darstellung von Umgebungen und formalisiert nicht-strikte Kontrolle durch Auszeichnung von speziellen Kontexten ("Evaluation Contexts"). Desweiteren enthält [AFM+95] einen Beweis für eine Komplexitätsabschätzung, nämlich, daß zu jeder nicht-strikten Berechnung ohne "Sharing" eine höchstens genauso lange mit "Sharing" existiert.

Auch die vorliegende Arbeit enthält eine Ein-Schritt-Semantik für nicht-strikte funktionale Berechnung mit "Sharing". Diese wurde zum ersten Mal im Rahmen des γ-Kalküls [Smo94c] veröffentlicht, dort aber nicht formal untersucht. Umgebungen oder *let*-Ausdrücke lassen sich im δ-Kalkül mittels paralleler Komposition und Deklaration darstellen. Im Gegensatz zu allen anderen Lösungen bilden wir hier nicht-strikte Kontrolle auf nebenläufige Kontrolle im δ-Kalkül selbst ab. Unsere Einbettung des nicht-strikten λ-Kalküls in den δ-Kalkül führt "Sharing" von funktionalen Argumenten ein, so daß Terminierung erhalten bleibt, aber möglicherweise Berechnungsschritte eingespart werden. Obwohl wir diese Eigenschaft hier nicht beweisen, ist der Autor davon überzeugt, daß die hier vorgestellten Methoden auch zu diesem Zweck hinreichend sind.

1.4.2 Funktionale als nebenläufige Berechnung

Im Kontext von Verteilung und Parallelisierung von funktionalen Sprachen sind eine Vielzahl von Modellen für nebenläufige Berechnung entwickelt worden [LTLG92]. Dabei gibt es im wesentlichen zwei Möglichkeiten, nämlich ein funktionales Modell mit nebenläufigen Konzepten anzureichern, wie zum Beispiel in [BMT92, TLP+93, Nik94], oder aber mit einem nebenläufigen Modell zu starten und darin funktionale Berechnung einzubetten [Bou89, Tho89, MPW92, Smo94c].

Diese Arbeit zählt zu den Vertretern des zuletzt genannten Ansatzes, die sich wiederum in zwei Gruppen einteilen lassen. Die erste Gruppe verwendet komplexe Substitutionen von Variablen durch Ausdrücke [Bou89, Tho89]. Die zweite Gruppe benutzt primitive Substitutionen [MPW92, Smo94c], welche Variablen durch Variablen oder Namen durch Namen ersetzen. Davide Sangiorgi zeigt in [San92], daß die Expressivität von beiden Ansätzen übereinstimmt. Der δ-Kalkül ist wiederum ein Vertreter des letzteren Ansatzes, was im Hinblick auf Komplexität notwendig ist.

Ein Vergleich von δ-, γ- und π-Kalkül. Die vorangehende Klassifizierung läßt den korrekten Schluß zu, daß der π-Kalkül [Mil91, MPW92, EN86], der γ-Kalkül [Smo94c] und der δ-Kalkül eng verwandt sind. Dennoch unterscheiden sich die Ziele des δ- und γ-Kalküls deutlich von denen des π-Kalküls. Der π-Kalkül modelliert

kommunizierende Prozesse, mit Schwerpunkt auf Synchronisation und Indeterminismus, während der γ-Kalkül als einheitliches Modell für funktionale, nebenläufige und objekt-orientierte Berechnung konzipiert ist. Der δ-Kalkül ist im wesentlichen eine Einschränkung des γ-Kalküls auf uniform nebenläufige Berechnung.

Formal unterscheiden sich δ, γ und π allerdings nicht allzu sehr. Dabei beziehen wir uns auf neuere, reduktionsbasierte Darstellungen von π [Mil91, Bou92, HY93] und nicht etwa auf ältere, die annotierte Transitionssysteme benutzen [MPW92, Mil90, Mil89]. Die Basiskombinatoren aller drei Kalküle sind parallele Komposition und Deklaration, deren Eigenschaften sich besonders einfach mittels einer strukturellen Kongruenz formalisieren lassen. Die Idee stammt vom π-Kalkül [Mil91], ist von der "chemical abstract Machine" [BB90] motiviert und wurde für δ und γ übernommen.

Der Vergleich zwischen δ, γ und π läßt sich am leichtesten in einem weiteren Kalkül führen, der alle drei Kalküle umfaßt und in [Smo94c] unter dem Namen κ-Kalkül vorgestellt wurde. Die Ausdrücke von κ sind diejenigen von δ, erweitert um Einmal-Abstraktionen $x{::}\overline{y}/E$.

$$E, F \;::=\; x{::}\overline{y}/E \mid x{:}\overline{y}/E \mid x\,\overline{y} \mid x = y \mid E \wedge F \mid \exists x\, E$$

In der Kongruenz von κ verhalten sich Abstraktionen wie replizierte Einmal-Abstraktionen.

$$x{:}\overline{y}/E \;\equiv\; x{:}\overline{y}/E \wedge x{::}\overline{y}/E$$

Anstelle von Applikation (A) erlaubt κ Kanalkommunikation (C), hier über den Kanal x:

$$x{::}\overline{y}/E \wedge x\,\overline{z} \;\rightarrow_C\; E[\overline{z}/\overline{y}]$$

In der Tat läßt sich Applikation (A) durch Replikation und Kanalkommunikation (C) ausdrücken:

$$x{:}\overline{y}/E \wedge x\,\overline{z} \;\equiv\; x{:}\overline{y}/E \wedge x{::}\overline{y}/E \wedge x\,\overline{z} \;\rightarrow_C\; x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}]$$

Desweiteren gehört Elimination (E) zur Reduktion von κ, in der gleichen Form wie für δ. Logische Variablen vermöge Gleichungen und Elimination sind in π nicht vorhanden, so daß wir κ als Erweiterung von π um logische Variablen interpretieren können.

Der γ-Kalkül ist eine konservative Erweiterung von δ um Zellen, welche Uniformität und Konfluenz im allgemeinen nicht erhalten. Eine Zelle $x{:}\,y$ besteht aus einem persistenten Namen x und einer veränderbaren Referenz y auf den Inhalt der Zelle. Ein Prozeß kann auf den Inhalt einer Zelle zugreifen, wenn er die alte Referenz u durch eine neue Referenz v ersetzt.

$$x{:}\,y \wedge x\,u\,v \;\rightarrow_{Cell}\; x{:}\,v \wedge u = y$$

Der aktuelle Inhalt einer Zelle hängt also davon ab, mit welchen Prozessen eine Zelle zuvor kommuniziert hat, ist also nicht unabhängig von der Berechnungsreihenfolge. Zum Beispiel reduziert $x{:}\,0 \wedge x\,y\,1 \wedge x\,z\,2$ zu $x{:}\,2 \wedge y = 0 \wedge z = 1$ und zu $x{:}\,1 \wedge y = 2 \wedge z = 0$.

In κ lassen sich Zellen unter Verwendung von Kanalkommunikation simulieren.

$$x{:}\,y \;=\; \exists z\,(x{::}u\,v/(u = y \wedge z\,v\,) \wedge z{:}w/x{::}u\,v/(u = w \wedge z\,v\,))$$

Zellen sind von imperativer Natur, ähnlich zu Zuweisungen, und ermöglichen die Darstellung des Zustandes von nebenläufigen Objekten [Smo94c, HSW95]. Aber auch im π-Kalkül ist nebenläufige objekt-orientierte Programmierung darstellbar [Wal91, VT93, Vas94b, Vas94a]. Alternative Mechanismen zu Kanälen oder Zellen bieten "Ports" [JMH93] im Kontext von nebenläufiger Berechnung mit Constraints und "M-Structures" [BNA91] im Kontext von paralleler funktionaler Berechnung. In der zuletzt genannten Arbeit werden auch logische Variablen in eine funktionale Sprache integriert ("I-Structures"), wobei Konfluenz verloren geht, weil dort die Ungebundenheit einer Variable wie logische Information behandelt wird.

Funktionale Berechnung im π-Kalkül. Robin Milner hat Einbettungen des strikten und nicht-strikten λ-Kalküls in den π-Kalkül angegeben und als adäquat bewiesen [Mil92]; dadurch wurde die vorliegende Arbeit maßgeblich beeinflußt. Die Einbettungen in dieser Arbeit wurden in [Smo94c] im Rahmen des γ-Kalküls vorgeschlagen, aber dort nicht formal untersucht.

Milners Einbettung des strikten λ-Kalküls in den π-Kalkül verbietet die Applikation von Funktionen mit noch nicht reduzierten Argumenten, im Gegensatz zu der hier vorgestellten Einbettung in den δ-Kalkül. Diese Restriktion ist für Milners Beweise notwendig, aber nicht von prinzipieller Natur wie in [OD93] ohne Beweis illustriert wird. Denn Milners Beweise beruhen darauf, daß eine Bijektion zwischen Berechnungen in beiden Kalkülen existiert. Im Falle des δ-Kalküls reicht eine Injektion aus, denn wegen uniformer Konfluenz verhalten sich alle Berechnungen des gleichen Ausdruckes gleich.

Milners Einbettung des nicht-strikten λ-Kalküls ist besonders einfach. Allerdings erhält sie das Komplexitätsverhalten des nicht-strikten λ-Kalküls. Dies hat wiederum beweistechnische Gründe, denn eine Einbettung mit dem gewünschten Komplexitätsverhalten existiert wohl [OD93].

1.4.3 Optimale Reduktion, Termersetzung und Typsysteme

Der δ-Kalkül stellt sicher, daß funktionale Argumente stets höchstens einmal ausgewertet werden. Dennoch ist seine Reduktion nicht optimal im Sinne von Lévi [GAL92, Lam90, Kat90]. Dies liegt an unserer Einschränkung auf schwache Reduktion, also dem Verbot von Reduktion in Abstraktionen. Steht zum Beispiel *Deep* für $\lambda z.Iz$, dann hat die folgende Ableitung im strikten λ-Kalkül die Länge 5:

$$(\lambda y.y\,(yz))\,Deep \;\to\; Deep\,(Deep\,z) \;\to\; Deep\,(Iz) \;\to\; Deep\,z \;\to\; Iz \;\to\; z$$

Einen Schritt könnten wir durch Reduktion unterhalb von Abstraktion einsparen indem wir *Deep* zu Beginn zu $\lambda z.z$ simplifizieren.

Reduktion im δ-Kalkül ist ähnlich definiert, wie diejenige von Termersetzungssystemen modulo Kongruenz. Deren allgemeine Theorie [DJ90] läßt sich allerdings nicht direkt anwenden, da die Axiome des δ-Kalküls mit Substitutionen und Nebenbedingungen über gebundene Variablen versehen sind.

Der Konfluenzbeweis des δ-Kalküls beruht auf einem lokalen Kriterium, nämlich uniformer Konfluenz. Dieses ist eine Verschärfung von starker Konfluenz, welche für die Termersetzungssysteme bekannt ist [DJ90, Hue80]. Der entscheidende Unterschied besteht darin, daß uniforme Konfluenz Uniformität im Bezug auf Komplexität und Terminierung erzwingt.

Es ist bemerkenswert, daß funktionale Kalküle mit schwacher Reduktion einfache Konfluenzbeweise zulassen, die ohne Terminierungsbetrachtungen auskommen, welche bei Verwendung von lokaler Konfluenz notwendig sind. Dadurch unterscheiden sich unsere Konfluenzbeweise von denjenigen für tiefe λ-Kalküle. Diese verwenden zum Beispiel die Methode der endlichen Entwicklungen ("finite Developments") [Bar84, ORH93a].

Typsysteme für den δ-Kalkül betrachten wir in dieser Arbeit nicht. Aber auch zu diesem Thema existieren bereits erste Resultate wie zum Beispiel das polymorphe Typsystem in [MS95, Mül94]. Dabei tritt im Vergleich zum λ-Kalkül das Problem auf, daß Eingabe und Ausgabe im δ-Kalkül nicht statisch determiniert sind. In [MN95] wird ein Implementierungskonzept für Meta-Eigenschaften wie Typinferenz oder abstrakte Interpretation vorgeschlagen, das auf tiefer Reduktion basiert.

1.4.4 Nebenläufige Programmierung mit Constraints

Eine Wurzel des δ-Kalküls ist die nebenläufige Programmierung mit Constraints ("concurrent constraint Programming"). Der δ-Kalkül hat mit dieser relationale Schreibweisen, Gleichheit erster Ordnung, logische Variablen, parallele Komposition und Deklaration gemein. Andere Aspekte, wie zum Beispiel Suche im Falle

von disjunktiver Information, Constraintsimplifikation und Indeterminismus durch "committed Choice", klammert er aus. Eine einheitliche formale Beschreibung all dieser Aspekte liefert der Oz-Kalkül [Smo94a, Smo94b], der sowohl δ als auch γ erweitert und als Fundierung der nebenläufigen Constraintsprache Oz [Smo94d, HM94, MMPS94] dient.

Historischer Abriß. Nebenläufige Programmierung mit Constraints verallgemeinert sowohl nebenläufige logische Programmierung ("concurrent logic Programming"), als auch logische Programmierung mit Constraints ("constraint logic Programming").

Logische Programmierung hat sich aus dem automatischen Beweisen basierend auf Resolution entwickelt (siehe z.B. [Fit90]). Prolog ist die bekannteste und älteste logische Programmiersprache (siehe z.B. [SS86, O'K90]). Sie wurde 1973 unabhängig von Alan Colmerauer und Robert Kowalski konzipiert und hat wesentlich zur Etablierung von Logik erster Stufe als Grundlage von Berechnung beigetragen. Notwendig dazu waren D.H.D. Warrens Implementierungstechniken [War80, AK91], mit denen sich die Effizienz von Prolog nachweisen ließ. Diese Techniken verwenden "Backtracking" zur Realisierung von Suche.

Logische Programmierung mit Constraints wurde mit Prolog II [CKvC83] aus der Taufe gehoben. Dabei sind Constraints prädikatenlogische Formeln erster Stufe, die zur abstrakten Beschreibung von logischen Datenstrukturen dienen, typischerweise von arithmetischen Werten oder von verschiedenartigen Bäumen [ST92, AKPS92, NPT93, NP93, Frü93]. Ein vereinheitlichendes Berechnungsmodell ("CLP-scheme") stellten Joxan Jaffar und Jean-Louis Lassez in [JL87] vor. Weitere Darstellungen werden durch [JM94, HS88] referiert. Zu den Sprachen dieser Sparte gehören Prolog III [Col90], Chip [DSVH90], clp(FD) [DC93], cc(FD) [vHSD93] und im weiteren Sinne auch Life [AKDM+94, AK93].

Nebenläufige logische Programmierung begann mit [CG81] und wurde unter anderem von Ehud Sharpiro [Sha87] und vom japanischen "fifth Generation Project" propagiert.

Nebenläufige Programmierung mit Constraints kombiniert logische Programmierung mit Constraints und nebenläufige logische Programmierung. Vijay Saraswat [Sar89, SR90, Sar91] schlug ein erstes einheitliches Modell vor, welches in der Zwischenzeit intensiv untersucht wurde [dBP92, dBKP92]. Neuere nebenläufige Sprachen mit Constraints sind AKL [Jan94, JH91] und Oz [Smo94d]. Eine ausführlichere Darstellung des Gebietes mit weitreichenden Vergleichen befindet sich in [Jan94].

Nebenläufige Programmierung mit Constraints. Der Schwerpunkt von nebenläufigen Sprachen mit Constraints liegt auf nebenläufigen und inkrementellen

Aspekten von Berechnung mit partieller, logischer Information. Ihre Modelle basieren auf der Metapher eines Berechnungsraumes, welcher aus Aktoren und Constraintspeicher besteht. Die Aktoren können auf die Information des Constraintspeichers zugreifen und, falls genügend Information vorhanden ist, reduzieren. Ein Aktor verschwindet bei Reduktion, fügt aber möglicherweise neue Information und Aktoren hinzu.

Ein typischer Aktor ist ein relationales Konditional, das im Gegensatz zu einem funktionalen kein Resultat zurückgibt, sondern neue Berechnungen freisetzt. Wird die Bedingung φ eines relationalen Konditionals

$$\mathsf{if}\ \varphi\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi}$$

vom Constraintspeicher logisch impliziert ("Entailment"), so kann das Konditional zum then-Zweig E reduziert werden; ist φ im Kontext des Constraintspeichers unerfüllbar ("Disentailment"), so kann es zum else-Zweig F reduziert werden. Anderenfalls muß das Konditional darauf suspendieren, daß genügend Information von anderen Aktoren angehäuft worden ist, um Gültigkeit oder Unerfüllbarkeit zu entscheiden. Die Idee, die Reduktionsbedingung eines Konditionals logisch zu erklären, stammt von Michael Maher [Mah87].

Auch Applikationen lassen sich als Aktoren auffassen. Im Falle von Prozeduren erster Stufe sind Prädikatsdefinitionen statisch gegeben, wodurch Prädikatsapplikationen stets reduzierbar sind. Im Falle von Prozeduren höherer Ordnung werden Prozeduren dynamisch erzeugt, so daß Applikationen analog zu Konditionalen suspendieren können [NS94]. Aus diesem Grund können wir für die Zwecke des δ-Kalküls auf Konditionale verzichten. Dadurch verlieren wir aber die Expressivität zur Darstellung von Verbunden ("Records") mit Adjunktion im Stile von [Smo94c].

Eine Prozedur erster Ordnung wird zum Beispiel durch die Abstraktion

$$square{:}x\, y/y = x * x$$

beschrieben. Diese können wir auch als Prädikatsdefinition auffassen:

$$\forall x \forall y\, (square(x, y) \ \leftrightarrow\ y = x * x)\,.$$

Nun macht eine prädikatenlogische Interpretation Sinn. Dazu würden wir *square* als Prädikatssymbol auffassen, das die Menge aller Paare $(x,\ y)$ denotiert, die den Constraint $y = x * x$ erfüllen.

Der δ-Kalkül als Constraintsprache. Wir können uns auf den Standpunkt stellen, daß der δ-Kalkül implizit triviale Constraints φ enthält und daß seine Variablen Namen von Abstraktionen denotieren.

$$\varphi\ ::=\ x = y\ \mid\ \varphi_1 \wedge \varphi_2\ \mid\ \exists x \varphi$$

In dieser Sichtweise integriert der δ-Kalkül Constraints erster Ordnung mit Prozeduren höherer Ordnung. Die Tatsache, daß sich diese Constraints durch Elimination (E) simplifizieren lassen, ist einer der wesentlichen Gründe für die Einfachheit des δ-Kalküls.

Wir können jeden Ausdruck des δ-Kalküls als Berechnungsraum interpretieren, dessen Information aus benannten Abstraktionen und Gleichungen besteht und dessen Aktoren Applikationen sind. In Analogie zu Konditionalen suspendieren Applikationen nun, falls nicht genügend Information vorhanden ist, also wenn noch keine passende Abstraktion existiert.

Der ρ-Kalkül [NS94] ist eine Variante des δ-Kalküls, welcher auch formal Prozeduren höherer Ordnung und Constraints erster Ordnung kombiniert. Er ist durch ein Constraintsystem parametrisiert und unterscheidet zwischen Namen und Variablen explizit. Jede seiner Prozeduren ist mit einem eindeutigen Namen versehen und mit jeder dynamisch erzeugten Prozedur wird ein neuer Name generiert. Dazu verwendet der ρ-Kalkül einen Mechanismus, der auf der Deklaration von Namen beruht und unabhängig in [HSW93, PS93, Ode94] vorgestellt wurde.

In der Tat ist auch der ρ-Kalkül uniform konfluent. Dies ist allerdings nicht einfach zu beweisen [NS94] und setzt eine geeignete Behandlung von Inkonsistenzen voraus. Im Gegensatz zum ρ-Kalkül vermeidet der δ-Kalkül explizite Namen und logische Inkonsistenzen. Dies ist ein weiterer Grund für seine Einfachheit.

Die Verwendung von Constraints höherer Ordnung anstelle von Constraints erster Ordnung führt zu Unifikation höherer Ordnung. Der hier vorgestellte Ansatz vermeidet die damit verbundene Problematik, wodurch er sich zum Beispiel von λ-Prolog [NM88] unterscheidet.

Die Modellierung von enkapsulierter Suche. Suche ist ein zentraler Aspekt von nebenläufiger Constraintprogrammierung, wie zum Beispiel die Sprachen AKL [JH91, Jan94] und Oz [Smo94d, MPSW94] belegen. Das Ziel dabei ist disjunktive Information zu verarbeiten, durch welche sich Probleme mehreren Lösungen spezifizieren lassen. Je nach Anwendung ist man an der Suche nach einer oder nach allen Lösungen interessiert, oder aber am Testen einer vorgegebenen Lösung. Idealerweise sollte eine Problemspezifikation für verschiedenen Lösungsverfahren einsetzbar sein. So ließe sich zum Beispiel eine Grammatik sowohl zum Generieren der von ihr beschriebenen Wörter als auch zum Testen verwenden.

Möchte man nebenläufig verschiedene Alternativen durchsuchen, reicht das Konzept von "Backtracking" nicht aus. Auch die Metapher eines einzigen Berechnungsraumes dient nur zur Beschreibung einer einzigen Alternative. Um mehrere Alternativen nebenläufig zu explorieren, muß deren spekulative Information voneinander getrennt bleiben, also jede Alternative in einem eigenen Berechnungsraum abgear-

beitet werden. Dies führt zu einem System von ineinander geschachtelten Berechnungsräumen, wobei Information stets nach innen hin sichtbar ist. Mit dem gleichen Konzept läßt sich Suche enkapsulieren [JH91, SS94, SSW94], wodurch sich Suchprobleme als Teile von größeren Aufgaben integrieren lassen. Zur Enkapsulierung genügt es, globale sichere Information von lokaler spekulativer zu trennen, also Suche in einem lokalen Berechnungsraum ablaufen zu lassen.

Eine interessante Frage im Kontext von Suche ist, wie sich der Aufbau von Alternativen dynamisch kontrollieren läßt. Dies ist in Oz [SSW94, SS94] mit folgender Idee gelöst: Alternativen werden nach dem Erzeugen nicht automatisch exploriert, sondern in abstrahierter Form ("eingepackt") zurückgegeben. Der Programmierer kann deren Verwaltung gestalten, indem er die Alternative seiner Wahl in einem lokalen Berechnungsraum appliziert ("auspackt"). Zu diesem Zweck ist die nebenläufige Kontrolle des δ-Kalküls wiederum nützlich, denn sie erlaubt Suche dynamisch und durch Bedarf zu steuern.

1.5 Gliederung der Arbeit

In Kapitel 2 diskutieren wir allgemeine Eigenschaften von Berechnungskalkülen. Der Schwerpunkt liegt auf Komplexität und der Adäquatheit von Kalküleinbettungen. In Kapitel 3 führen wir den δ-Kalkül ein und beweisen seine uniforme Konfluenz, bis auf Details im Zusammenhang mit α-Standardisierung und Pränexnormalformen, die wir in Kapitel 5 nachholen. In Kapitel 4 betten wir den strikten und nicht-strikten λ-Kalkül in den δ-Kalkül ein und beweisen im ersteren Fall die Adäquatheit.

Kapitel 2

Berechnungskalküle

Wir führen Berechnungskalküle formal ein und diskutieren Konfluenzbegriffe und Komplexitätsmaße, sowie Terminierung und algebraische Operationen auf Kalkülen. Wir definieren Kalküleinbettungen und stellen allgemeine Kriterien für deren Adäquatheit bezüglich gegebener Komplexitätsmaße vor. Desweiteren führen wir Basen als Teilsysteme von Kalkülen ein, die in dem Sinne vollständig sind, daß sie wichtige Eigenschaften des zugrundeliegenden Kalküls determinieren.

2.1 Präliminarien

In diesem Kapitel denotieren K, L, M Mengen mit Elementen $\kappa \in K$, $\lambda \in L$ und $\mu \in M$. Die Mengenoperationen bezeichnen wir wir üblich, $K \cup L$ für die Vereinigung von K und L und $K \cap L$ für deren Schnitt . Das Symbol $\emptyset$ steht für die leere Menge, $K \subseteq L$ für die Inklusion von K in L, und $K \times L$ für deren kartesisches Produkt. Wir nennen K und L disjunkt, falls $K \cap L = \emptyset$ gilt. Eine Relation auf K ist eine Teilmenge von K und eine binäre Relation auf K eine Teilmenge von $K \times K$.

Vereinigung, Schnitt und Inklusion von Relationen führen wir auf die analogen Begriffe auf Mengen zurück. Die Komposition $(\rightarrow_1 \circ \rightarrow_2)$ zweier Relationen $\rightarrow_1$ auf $K \times L$ und $\rightarrow_2$ auf $L \times M$ ist eine Relation auf $K \times M$. Diese ist definiert durch $\kappa \ (\rightarrow_1 \circ \rightarrow_2) \ \mu$ genau dann, wenn $\lambda \in L$ existiert mit $\kappa \rightarrow_1 \lambda$ und $\lambda \rightarrow_2 \mu$. Sei $\rightarrow$ eine Relationen auf $K \times L$, $K' \subseteq K$ und $L' \subseteq L$. Das Inverse von $\rightarrow$ ist eine Relation $\leftarrow$ auf $L \times K$, wobei $\lambda \leftarrow \kappa$ genau dann gilt, wenn $\kappa \rightarrow \lambda$ erfüllt ist. Gelegentlich bezeichnen wir die inverse Relation von $\rightarrow$ auch mit $\rightarrow^{-1}$. Die Einschränkung von $\rightarrow$ auf $K' \times L'$ ist definiert durch $\rightarrow_{|(K' \times L')} = \ (\rightarrow \ \cap (K' \times L'))$. Stimmen die Mengen K und L überein, dann heißt die Menge K' abgeschlossen unter $\rightarrow$, falls $\rightarrow_{|K' \times K} \subseteq \ (K' \times K')$.

Seien nun $\rightarrow_K$ eine binäre Relation auf K, $\rightarrow$ eine Relation auf $K \times L$ und $\rightarrow_L$ eine

binäre Relation auf L. Dann nennen wir $\rightarrow$ linksinvariant unter $\rightarrow_K$, falls $(\rightarrow_K \circ \rightarrow) \subseteq \rightarrow$ und rechtsinvariant unter $\rightarrow_L$, falls $(\rightarrow \circ \rightarrow_L) \subseteq \rightarrow$. Stimmen K und L überein, dann nennen wir $\rightarrow$ invariant unter $\rightarrow_L$, falls $\rightarrow$ rechts- und linksinvariant unter $\rightarrow_L$ ist.

Eine Äquivalenzrelation $\equiv$ auf K ist eine binäre Relation auf K, die reflexiv, $=_K \subseteq \equiv$, symmetrisch, $\equiv^{-1} \subseteq \equiv$, und transitiv, $(\leq \circ \leq) \subseteq \leq$, ist. Äquivalenzrelationen bezeichnen wir mit $\equiv$, wobei wir gegebenenfalls mit der zugrundeliegenden Menge indizieren und dann $\equiv_K$, $\equiv_L$ bzw $\equiv_M$ schreiben.

Unter einer (partiellen) Ordnung $\leq$ auf einer Menge K verstehen wir eine binäre Relation auf K, die transitiv und antisymmetrisch ist, also $(\leq \cap \geq) \subseteq =_K$, wobei $=_K$ die Gleichheitsrelation auf K denotiert. Eine obere (bzw untere) Schranke einer Menge $K' \subseteq K$ in einer Ordnung $\leq$ auf K ist ein Element $\kappa \in K$ mit $\kappa' \leq \kappa$ (bzw $\kappa \leq \kappa'$) für alle $\kappa' \in K'$. Eine Abbildung γ zwischen zwei Mengen mit partiellen Ordnungen $(K, \leq_K)$ und $(L, \leq_L)$ heißt monoton, falls mit $\kappa_1 \leq_K \kappa_2$ auch $\gamma(\kappa_1) \leq_L \gamma(\kappa_2)$ gilt.

Natürliche Zahlen, stets inklusive 0, bezeichnen wir mit i, j und m, n. Die Menge aller natürlichen Zahlen ist durch $\leq$ und $<$ geordnet, genauso wie die Menge aller natürlichen Zahlen mit unendlich ∞. Unter einer endliche Folge $(\kappa_i)_{i=0}^n$ über K verstehen wir eine Abbildung von $\{0, \ldots, n\}$ nach K, die wir alternativ durch $(\kappa_0, \ldots, \kappa_n)$ denotieren. Eine unendliche Folge $(\kappa_i)_{i=0}^\infty$ über K ist eine Abbildung von den natürlichen Zahlen nach K. Häufig schreiben wir $(\kappa_i)_i$ für eine endliche oder unendliche Folge, wobei wir die Indexgrenzen weglassen.

Eine binäre Relation $<$ auf K heißt wohlfundiert, wenn es keine unendliche absteigende Kette bezüglich $<$ gibt, also keine Folge $(\kappa_i)_{i=0}^\infty$ existiert, mit $\kappa_{i+1} < \kappa_i$ für alle $i \geq 0$. Wir werden wohlfundierte Induktion, d.h. Induktion über wohlfundierte Relationen, frei verwenden (siehe zum Beispiel [Win93]). Der transitive Abschluß jeder wohlfundierten Relation ist eine Ordnung.

2.2 Kalküle und Berechnungen

Unter einem Kalkül verstehen wir stets einen Berechnungskalkül der folgenden Bauart.

Definition 2.2.1 *Ein* Kalkül *ist ein Tripel $\mathcal{K} = (K, \equiv, \rightarrow)$ bestehend aus einer Menge K, einer Äquivalenzrelation $\equiv$ auf K, und einer binären Relation $\rightarrow$ auf K, die invariant unter $\equiv$ ist.*

Die Elemente von K heißen Ausdrücke des Kalküls, $\equiv$ ist seine Kongruenz und $\rightarrow$ seine Reduktion. Die Invarianz von $\rightarrow$ unter $\equiv$, also die Inklusion $(\equiv \circ \rightarrow \circ \equiv) \subseteq$

$\rightarrow$, nennen wir Kalküleigenschaft.

In der Definition eines Kalküls übernimmt die Kongruenz die Rolle von Gleichheit auf der Menge der Ausdrücke. Diese Unterscheidung ist notwendig, da zum Beispiel in δ, γ und π Kongruenzen natürlich auftreten, die nicht mit der Gleichheit auf Ausdrücken übereinstimmen. Natürlich könnten wir alternativ stets zu Kongruenzklassen übergehen und mit der Gleichheit solcher Klassen rechnen, aber dies würde mit Sicherheit schwerfällig. Insbesondere werden wir Kongruenzen verwenden, um spezielle Reduktionsschritte nach Bedarf auszublenden, indem wir sie zur Kongruenz hinzuschlagen.

Weitere prototypische Beispiele für Kalküle im obigen Sinne sind der reine, schwache, strikte und nicht-strikte λ-Kalkül (vgl. Kapitel 4). Auch für diese sind Kongruenzen sinnvoll, denn sie erlauben von α-Konversion zu abstrahieren. Jedes Termersetzungssystem [DJ90] mit Signatur Σ und Regelmenge R definiert einen Kalkül $(\mathcal{T}_\Sigma, =, \rightarrow_R)$, wobei $\mathcal{T}_\Sigma$ die Menge aller Σ-Terme, $=$ die Gleichheit solcher Terme und $\rightarrow_R$ die von den Regeln erzeugte Ableitungsrelation ist. Auch ein Termersetzungssystem modulo Kongruenz $(\Sigma, \equiv, R)$ im Stile von [DJ90][1] definiert einen Kalkül $(\mathcal{T}_\Sigma, \equiv, (\equiv \circ \rightarrow_R \circ \equiv))$, wobei die Kongruenz des Termersetzungssystems mit derjenigen des Kalküls übereinstimmt.

Jede Turingmaschine [LP81, HU79] definiert einen Kalkül, dessen Ausdrücke die Konfigurationen der Turingmaschine sind. Die Kongruenz ist die Gleichheit auf Konfigurationen und die Reduktion stellt die Konfigurationsübergänge der Maschine dar.

Sei $\mathcal{K} = (K, \equiv, \rightarrow)$ ein Kalkül. Eine endliche Ableitung in $\mathcal{K}$ ist eine endliche Folge $(\kappa_0, \ldots, \kappa_n)$von Ausdrücken mit $\kappa_i \rightarrow \kappa_{i+1}$ für alle $0 \leq i \leq n-1$. Die Länge einer Ableitung $(\kappa_0, \ldots, \kappa_n)$ ist n. Eine unendliche Ableitung in $\mathcal{K}$ ist eine Folge $(\kappa_i)_{i=0}^{\infty}$ mit $\kappa_i \rightarrow \kappa_{i+1}$ für alle $0 \leq i < \infty$. Die Länge einer unendlichen Ableitung ist ∞. Mit $l_\mathcal{K}((\kappa_i)_i)$ bezeichnen wir die Länge einer endlichen oder unendlichen Abbleitung $(\kappa_i)_i$.

Die Menge aller Ableitungen in $\mathcal{K}$ ist partiell geordnet. Eine Ableitung $(\kappa_i)_i$ ist echt kleiner als eine Ableitung $(\kappa'_i)_i$, falls $l_\mathcal{K}((\kappa_i)_i) < l_\mathcal{K}((\kappa'_i)_i)$ und $\kappa'_j = \kappa_j$ für alle $0 \leq j \leq l_\mathcal{K}((\kappa'_i)_i)$. Ableitungen werden somit größer durch Anhängen und kleiner durch Abschneiden von Ableitungen.

Eine Berechnung in $\mathcal{K}$ ist eine maximale Ableitung in $\mathcal{K}$. Eine Berechnung eines Ausdruckes κ in $\mathcal{K}$ ist eine Berechnung, die mit κ beginnt, also eine Berechnung $(\kappa_i)_i$ mit $\kappa_0 \equiv \kappa$. Ein Ausdruck terminiert in $\mathcal{K}$, wenn all seine Berechnungen endlich sind.

Ein Ausdruck κ heißt irreduzibel in $\mathcal{K}$, falls kein κ' existiert mit $\kappa \rightarrow \kappa'$, also alle Berechnungen von κ die Länge 0 haben.

[1] Im Unterschied dazu betrachtet Gérard Huet [Hue80] die Relation $(\equiv \circ \rightarrow_R \circ \ldots \circ \rightarrow_R \circ \equiv)$.

Proposition 2.2.2 *Eine Ableitung ist genau dann eine Berechnung, wenn sie unendlich ist oder wenn sie endlich und ihr letztes Element irreduzibel ist. Jede Ableitung läßt sich zu einer Berechnung fortsetzen.*

Beweis. Offensichtlich. □

2.3 Uniforme Konfluenz

Im allgemeinen können Ausdrücke eines Kalküls sowohl endliche, als auch unendliche Berechnungen zulassen. Stellt man sich eine Berechnung als mögliche Ausführung eines Programmes vor, so läßt sich in solchen Kalkülen das Laufzeitverhalten von Programmen nur schwer oder gar nicht voraussagen.

Wir nennen einen Kalkül uniform, wenn für alle Ausdrücke κ des Kalküls alle Berechnungen von κ die gleiche Länge haben. Ein Kalkül $(K, \equiv, \rightarrow)$ heißt uniform konfluent, falls auf K die folgende Inklusion gilt:

$$(\leftarrow \circ \rightarrow) \subseteq (\equiv \cup (\rightarrow \circ \leftarrow)) .$$

Satz 2.3.1 *Jeder uniform konfluente Kalkül ist uniform.*

Uniforme Konfluenz ist also eine Eigenschaft von Kalkülen, die die Unabhängigkeit von Terminierung und Berechnungslänge von der Berechnungsreihenfolge sicherstellt.

Korollar 2.3.2 *Sei κ ein Ausdruck eines uniform konfluenten Kalküls. Existiert eine endliche Berechnung von κ, dann sind alle Berechnungen von κ endlich.*

Beweis. Ein Berechnung ist genau dann endlich, wenn ihre Länge endlich ist. Die Längen aller Berechnungen von κ stimmen nach Satz 2.3.1 überein. □

Beweis von Satz 2.3.1. Sei $(K, \equiv, \rightarrow)$ ein uniform konfluenter Kalkül. Wir zeigen mit Induktion über n, daß für alle κ aus K, die eine Berechnung der Länge n besitzen, alle Berechnungen von κ die Länge n haben[2].

Sei κ beliebig und $(\kappa_i)_i$ eine Berechnung der Länge n von κ. Im Falle $n = 0$ ist κ irreduzibel, so daß alle Berechnungen von κ die Form (κ_0) mit $\kappa_0 \equiv \kappa$ haben, also die Länge 0. Ist $n \geq 0$, dann betrachten wir eine zweite Berechnung $(\kappa'_i)_i$ von κ, also mit $\kappa'_0 \equiv \kappa \equiv \kappa_0$ und zeigen, daß auch diese die Länge n hat. Nun ist κ'_0 reduzierbar,

[2]Diese einfache Beweisidee stammt von Rolf Backofen.

so daß die Länge von $(\kappa_i')_i$ wegen Maximalität mindestens 1 sein muß. Es gibt also κ_1' mit $\kappa_1' \leftarrow \kappa \rightarrow \kappa_1$, so daß uniforme Konfluenz genau zwei Möglichkeiten zuläßt. Im Fall $\kappa_1' \equiv \kappa_1$ können wir die Induktionsvoraussetzung auf κ_1 anwenden. Somit hat die Berechnung $(\kappa_i')_{i\geq 1}$ die Länge $n-1$ und die Berechnung $(\kappa_i')_{i\geq 0}$ die Länge n. Im zweiten Fall existiert $\tilde{\kappa}_2$ mit $\kappa_1' \rightarrow \tilde{\kappa}_2 \leftarrow \kappa_1$. Sei $(\tilde{\kappa}_i)_{i\geq 2}$ eine beliebige Berechnung von $\tilde{\kappa_2}$, welche nach Proposition 2.2.2 existiert. Dann ist $(\kappa_1, \tilde{\kappa}_2, \ldots)$ eine Berechnung von κ_1, die nach Induktionsvoraussetzung die Länge $n-1$ hat. Somit existiert eine Berechnung der Länge $n-1$ von κ_1', nämlich $(\kappa_1', \tilde{\kappa}_2, \ldots)$. Aus der Induktionsvoraussetzung angewendet auf κ_1' folgt, daß die Länge von $(\kappa_i')_{i\geq 1}$ ebenfalls $n-1$ ist. Somit ist auch im zweiten Fall die Länge von $(\kappa_i')_{i\geq 0}$ gleich n. □

2.4 Konfluenz und Terminierung

In diesem Abschnitt übertragen wir die üblichen Konfluenzbegriffe [DJ90, Hue80] auf Kalküle und vergleichen sie mit dem Begriff der uniformen Konfluenz. Desweiteren übertragen wir den Begriff der Terminierung, wobei keinerlei Überraschungen auftreten.

Sei $\mathcal{K} = (K, \equiv, \rightarrow)$ ein beliebiger Kalkül. Dann definieren wir die folgenden Relationen, die alle sowohl von der Reduktion, als auch von der Kongruenz abhängen.

$$\rightarrow^0 = \equiv, \qquad \rightarrow^{n+1} = (\rightarrow^n \circ \rightarrow), \qquad \rightarrow^{\geq n} = \cup_{i=n}^{\infty} \rightarrow^i$$
$$\rightarrow^* = \rightarrow^{\geq 0} \qquad \rightarrow^{\leq n} = \cup_{i=0}^{n} \rightarrow^i, \qquad \rightarrow^{\epsilon} = \rightarrow^{\leq 1}.$$

Lemma 2.4.1 *Die Relation $\rightarrow^*$ ist die kleinste transitive Relation, die $\rightarrow$ und $\equiv$ enthält.*

Beweis. Wie im Falle der reflexiven transitiven Hülle. □

Wir nennen $\mathcal{K}$

lokal konfluent	*gdw.*	$(\leftarrow \circ \rightarrow) \subseteq (\rightarrow^* \circ {}^*\leftarrow)$,
konfluent	*gdw.*	$({}^*\leftarrow \circ \rightarrow^*) \subseteq (\rightarrow^* \circ {}^*\leftarrow)$,
Church-Rosser	*gdw.*	$(\leftarrow \cup \rightarrow)^* \subseteq (\rightarrow^* \circ {}^*\leftarrow)$,
stark konfluent	*gdw.*	$(\leftarrow \circ \rightarrow) \subseteq (\rightarrow^\epsilon \circ {}^\epsilon\leftarrow)$.

Wenn die Kongruenz von $\mathcal{K}$ mit der Gleichheit auf K übereinstimmt, fallen unsere Definitionen mit den üblichen zusammen [DJ90].

Proposition 2.4.2 *Uniforme Konfluenz impliziert starke Konfluenz und aus dieser folgt Konfluenz. Konfluenz ist äquivalent zur Church-Rosser-Eigenschaft und impliziert lokale Konfluenz.*

Beweis. Die einzige nicht triviale Aussage ist, daß Konfluenz aus starker Konfluenz folgt. Dies ist wohlbekannt, wenn die Kongruenz des Kalküls mit der Gleichheit auf Ausdücken übereinstimmt. Im allgemeinen ließe sich der Beweis durch Quotientenbildung auf diesen Fall zurückgeführen. Für einen direkten Beweis können wir, wie im Standardfall,

$$({}^{m}\!\leftarrow \circ \rightarrow^{n}) \subseteq (\rightarrow^{\leq n} \circ {}^{\leq m}\!\leftarrow)$$

mit simultaner Induktion über m und n nachweisen. □

Alle Konfluenzeigenschaften sind Vertauschungseigenschaften. Wir sagen wie üblich, daß zwei Relationen $\rightarrow_1$ und $\rightarrow_2$ vertauschen, falls sie

$$({}_{1}\!\leftarrow \circ \rightarrow_2) \subseteq (\rightarrow_2 \circ {}_{1}\!\leftarrow)$$

erfüllen. So ist zum Beispiel $\mathcal{K}$ genau dann konfluent, wenn $\rightarrow^*$ mit sich selbst vertauscht, und genau dann stark konfluent, wenn $\rightarrow^\epsilon$ mit sich selbst vertauscht. Lokale und uniforme Konfluenz sind Vertauschungseigenschaften in einem etwas weiteren Sinne.

$\mathcal{K}$ heißt terminierend, falls alle Berechnungen in $\mathcal{K}$ endlich sind, also jeder Ausdruck in K terminiert. Die Definition von Terminierung ist unabhängig von der Kongruenz von $\mathcal{K}$, da die Definition einer Berechnung in $\mathcal{K}$ unabhängig von dieser ist. Weiterhin sagen wir, daß eine binäre Relation $\rightarrow_1$ auf K terminiert, wenn der Kalkül $(K, =_K, \rightarrow_1)$ terminierend ist, wobei $=_K$ die Gleichheitsrelation auf K denotiert. Somit ist $\mathcal{K}$ terminierend genau dann, wenn seine Reduktion $\rightarrow$ terminiert, was wiederum äquivalent zur Wohlfundiertheit von $\leftarrow$ ist.

Proposition 2.4.3 *Jeder lokal konfluente, terminierende Kalkül ist konfluent.*

Beweis. Mit wohlfundierter Induktion über $\leftarrow$ (siehe zum Beispiel [Hue80]). □

2.5 Algebraische Operationen

Auf Kalkülen lassen sich die üblichen algebraischen Konstruktionen wie Einschränkung, Vereinigung, Quotient und kartesisches Produktes definieren. So ist zum Beispiel der δ-Kalkül die Einschränkung einer Kalkülvereinigung, aber auch die Vereinigung von Kalküleinschränkung. Denn Vereinigung kombiniert Applikation und Elimination und Einschränkung schließt Inkonsistenzen aus. Quotienten und kartesische Produkte berücksichtigen wir hier nicht, da wir diese im folgenden nicht benötigen.

Sei $\mathcal{K} = (K, \equiv, \rightarrow)$ ein Kalkül und $L \subseteq K$ eine Teilmenge von Ausdrücken, die unter Kongruenz und Reduktion abgeschlossen ist. Dann definieren wir die

Einschränkung von $\mathcal{K}$ auf L durch $(L, \equiv_{|(L\times L)}, \to_{|(L\times L)})$. Der Einfachheit halber lassen wir Einschränkungsoperatoren meist weg und schreiben kürzer $(L, \equiv, \to)$.

Proposition 2.5.1 *Einschränkungen von Kalkülen sind Kalküle.*

Beweis. Offensichtlich ist $\equiv_{|(L\times L)}$ eine Äquivalenzrelation auf L. Da L abgeschlossen unter Kongruenz und Reduktion ist, vererbt sich die Kalküleigenschaft:

$$(\equiv_{|(L\times L)} \circ \to_{|(L\times L)} \circ \equiv_{|(L\times L)}) \subseteq (\equiv \circ \to \circ \equiv)_{|(L\times L)} \subseteq \to_{|(L\times L)} . \qquad \square$$

Proposition 2.5.2 *Konfluenzeigenschaften bleiben bei Einschränkung erhalten.*

Beweis. Die einschränkende Menge ist abgeschlossen unter $\equiv$ und $\to$ und somit auch unter $\to^*$ und $\to^\epsilon$. $\square$

Die Vereinigung zweier Kalküle $(K, \equiv, \to_1)$ und $(K, \equiv, \to_2)$ ist ein Kalkül, den wir durch $(K, \equiv, (\to_1 \cup \to_2))$ definieren.

Proposition 2.5.3 *Seien $(K, \equiv, \to_1)$ und $(K, \equiv, \to_2)$ zwei uniform konfluente Kalküle und $\to = (\to_1 \cup \to_2)$ die Vereinigung von deren Reduktionen. Gilt weiterhin $({}_1\!\leftarrow \circ \to_2) \subseteq ((\to \circ \leftarrow) \cup \equiv)$, dann ist die Vereinigung der beiden Kalküle $(K, \equiv, \to)$ uniform konfluent.*

Beweis. Die folgenden Inklusionen folgen unmittelbar aus den Voraussetzungen:

$$\begin{aligned} (\leftarrow \circ \to) &= ({}_1\!\leftarrow \circ \to_1) \cup ({}_1\!\leftarrow \circ \to_2) \cup ({}_2\!\leftarrow \circ \to_1) \cup ({}_2\!\leftarrow \circ \to_2) \\ &\subseteq ((\to_1 \circ {}_1\!\leftarrow) \cup \equiv) \cup ((\to \circ \leftarrow) \cup \equiv) \cup ((\to_2 \circ {}_2\!\leftarrow) \cup \equiv) \\ &\subseteq ((\to \circ \leftarrow) \cup \equiv) \end{aligned}$$ $\square$

Nun können wir eine Eigenschaft zeigen, die als "Lemma von Hindley-Rosen" bekannt geworden ist. Dieses läßt sich unter anderem dazu verwenden, die Konfluenz von $(\beta\eta)$-Reduktion nachzuweisen (siehe zum Beispiel [Bar84], Seite 64).

Proposition 2.5.4 (Hindley-Rosen) *Seien $(K, \equiv, \to_1)$ und $(K, \equiv, \to_2)$ zwei konfluente Kalküle und $\to = (\to_1 \cup \to_2)$ die Vereinigung von deren Reduktionen. Dann gilt unter der Voraussetzung $({}_1^*\!\leftarrow \circ \to_2^*) \subseteq (\to^* \circ {}^*\!\leftarrow)$, daß die Vereinigung der beiden Kalküle $(K, \equiv, \to)$ konfluent ist.*

Beweis. Da die Relationen $\to_1$ und $\to_2$ konfluent sind, sind $\to_1^*$ und $\to_2^*$ uniform konfluent. Nach Proposition 2.5.3 ist $(\to_1^* \cup \to_2^*)$ uniform konfluent und somit nach Proposition 2.4.2 konfluent. Wegen $\to^* = (\to_1^* \cup \to_2^*)^*$ ist dann auch $\to$ konfluent. Die zuletzt ausgenutzte Gleichung gilt, da beide Seiten die kleinste Relation denotieren, die $\to_1$ und $\to_2$ enthält. $\square$

2.6 Komplexitätsmaße

Wir führen nun das Konzept eines Komplexitätsmaßes ein, um von der Länge von Berechnungen zu abstrahieren. Dies erlaubt uns Terminierung mit dem gleichen formalen Apparat zu beschreiben, wie Komplexität. Desweiteren wollen wir häufig nur ausgezeichnete Schritte einer Berechnung zählen, zum Beispiel nur die Applikationsschritte im Falle des δ-Kalküls, und benötigen dann Varianten des Längenmaßes.

Sei $\mathcal{K} = (K, \equiv, \to)$ ein Kalkül. Ein Komplexitätsmaß für $\mathcal{K}$ ist eine monotone Funktion, die Ableitungen in $\mathcal{K}$ auf natürliche Zahlen oder ∞ abbildet. Wir erinnern daran, daß die Menge aller Ableitungen eines Kalküls geordnet ist, wobei kürzere Ableitungen durch Abschneiden am Ende entstehen.

Ein wichtiges Komplexitätsmaß für diese Arbeit ist das Längenmaß $l_{\mathcal{K}}$. Ein weiteres ist das Terminierungsmaß $t_{\mathcal{K}}$, das für Ableitungen in $\mathcal{K}$ folgendermaßen definiert ist:

$$t_{\mathcal{K}}((\kappa_i)_i) = \begin{cases} 0 & \text{falls } l_{\mathcal{K}}((\kappa_i)_i) < \infty \\ \infty & \text{falls } l_{\mathcal{K}}((\kappa_i)_i) = \infty \end{cases}$$

Wir nennen ein Komplexitätsmaß uniform, falls es für alle Ausdrücke κ, eingeschränkt auf die Berechnungen von κ, konstant ist. Offensichtlich ist ein Kalkül genau dann uniform, wenn sein Längenmaß uniform ist.

Ist γ ein uniformes Komplexitätsmaß für $\mathcal{K}$, so definieren wir die Komplexität eines Ausdruckes κ durch

$$\gamma(\kappa) \;=\; \gamma((\kappa_i)_i) \qquad \text{für eine Berechnung } (\kappa_i)_i \text{ von } \kappa.$$

Ist γ jedoch nicht als uniform vorausgesetzt, so gibt es mindestens zwei Möglichkeiten, die Komplexität von Ausdrücken festzulegen, nämlich durch das Bilden von Infima oder Suprema. Wir definieren deshalb die Komplexität von Ausdrücken ausschließlich für uniforme Komplexitätsmaße.

Wir kommen nun zu einer Verfeinerung des Längenmaßes, das nur ausgezeichnete Schritte einer Berechnung zählt. Typischerweise treiben diese Schritte die eigentliche Berechnung voran (Applikation), wohingegen die anderen den erreichten Ausdruck standardisieren (Elimination).

Um diese Situation zu formalisieren, betrachten wir eine weitere binäre Relation $\to_r$ auf K, und definieren das r-Längenmaß $l^r_{\mathcal{K}}$ auf Ableitungen in $\mathcal{K}$ durch

$$l^r_{\mathcal{K}}((\kappa_i)_i) \;=\; \text{Anzahl}\,\{i \mid \kappa_i \to_r \kappa_{i+1}\}\,.$$

Natürlich ist $l^r_{\mathcal{K}}$ ein Komplexitätsmaß und gerade dieses Maß werden wir intensiv benutzen. Da wir an uniformer Berechnung interessiert sind, stellt sich nun die Frage, wann das r-Längenmaß uniform ist. Eine befriedigende Antwort können wir mit Hilfe des Begiffes einer Zerlegung ("Decomposition") angeben.

Definition 2.6.1 *Ein Paar* $\zeta = (\rightarrow_r, \rightarrow_c)$ *von binären Relationen auf* K *heißt* Zerlegung *für* $(K, \equiv)$*, falls die folgenden Eigenschaften gelten:*

(*Dec1*) *Das Tripel* $(K, \equiv, \rightarrow_r)$ *ist ein uniform konfluenter Kalkül.*

(*Dec2*) *Das Tripel* $(K, \equiv, \rightarrow_c)$ *ist ein konfluenter, terminierender Kalkül.*

(*Dec3*) *Die Relationen* $\rightarrow_r$ *und* $\rightarrow_c^*$ *vertauschen.*

Die Sprechweisen und Bezeichnungen in der Definition stammen daher, daß die Relation $(\rightarrow_r \cup \rightarrow_c)$ in eine Reduktion $\rightarrow_r$ und eine Kontrolle $\rightarrow_c$ zerlegt wird. Im Falle des δ-Kalküls ist zum Beispiel das Paar $(\rightarrow_A, \rightarrow_E)$ eine Zerlegung, d.h. Applikation läßt sich als eigentliche Reduktion des δ-Kalküls auffassen und Elimination lediglich als Kontrolle.

Wenn wir eine Zerlegung $\zeta = (\rightarrow_r, \rightarrow_c)$ für $(K, \equiv)$ betrachten, sind wir eigentlich an der Vereinigung $\mathcal{K}(\zeta) = (K, \equiv, (\rightarrow_r \cup \rightarrow_c))$ interessiert, möchten jedoch $\rightarrow_c$ Schritte ausblenden und lediglich $\rightarrow_r$ Schritte zählen. Es stellt sich somit die Frage, wann $l^r_{\mathcal{K}(\zeta)}$ uniform ist.

Um dieses Problem technisch zu handhaben, suchen wir nach einem Kalkül, der auch formal Kontrollschritte ausblendet. Dazu definieren wir $\overset{c}{\rightarrow}_r = (\rightarrow_c^* \circ \rightarrow_r \circ \rightarrow_c^*)$ und $\mathcal{K}^r_c = (K, \equiv, \overset{c}{\rightarrow}_r)$.

Satz 2.6.2 *Ist* $\zeta = (\rightarrow_r, \rightarrow_c)$ *eine Zerlegung für* $(K, \equiv)$*, dann sind* $l_{\mathcal{K}^r_c}$ *und* $l^r_{\mathcal{K}(\zeta)}$ *uniform. Desweiteren gilt* $l_{\mathcal{K}^r_c}(\kappa) = l^r_{\mathcal{K}(\zeta)}(\kappa)$ *für alle Ausdrücke* κ *von* K*.*

Die Uniformitätseigenschaft ist die zentrale Behauptung des Satzes. Aus dieser läßt sich $l_{\mathcal{K}^r_c}(\kappa) = l^r_{\mathcal{K}(\zeta)}(\kappa)$ folgern, wozu man zusätzlich die Terminierung von $\rightarrow_c$ benötigt.

Wir leiten nun den Beweis von Satz 2.6.2 durch einige Vorbemerkungen ein: Der Kalkül $\mathcal{K}(\zeta)$ ist zwar nach Proposition 2.5.4 konfluent, aber weder notwendigerweise uniform konfluent, noch uniform. Dies liegt daran, daß wir lediglich Konfluenz für $\rightarrow_c$ voraussetzen. Die Situation für den Kalkül $\mathcal{K}^r_c$ ist scheinbar noch widersinniger; denn wie im Satz behauptet, ist $\mathcal{K}^r_c$ uniform, aber im allgemeinen noch nicht einmal konfluent. Das Problem liegt darin, daß die Resultate von Berechnungen eines Ausdruckes in $\mathcal{K}^r_c$ nur bis auf ungerichtete Anwendungen von $\rightarrow_c$ übereinstimmen, aber nicht kongruent sind.

Diese Problematik von $\mathcal{K}^r_c$ läßt sich lösen, indem wir zu einer anderen Kongruenz übergehen, nämlich zu $\equiv_c = ({}_c\!\leftarrow \cup \rightarrow_c)^*$, und den Kalkül $\hat{\mathcal{K}}^r_c$ betrachten:

$$\hat{\mathcal{K}}^r_c \;=\; (K, \equiv_c, (\equiv_c \circ \rightarrow_r \circ \equiv_c))$$

Unter Rückgriff auf diese Hilfskonstruktion werden wir nun Satz 2.6.2 beweisen. Dazu benötigen wir drei vorbereitende Lemmata, für die wir die Voraussetzungen des Satzes annehmen.

Lemma 2.6.3 *$\hat{\mathcal{K}}_c^r$ ist ein uniform konfluenter Kalkül, der die Darstellung $\hat{\mathcal{K}}_c^r = (K, (\to_c^* \circ {}^*_c\!\leftarrow), ({}^c\!\to_r \circ {}^*_c\!\leftarrow))$ besitzt.*

Beweis. Offensichtlich ist $\hat{\mathcal{K}}_c^r$ ein Kalkül, d.h. $\equiv_c$ ist eine Äquivalenzrelation, und die Kalküleigenschaft gilt. Wegen Konfluenz (*Dec2*) ist $\to_c$ Church-Rosser und somit hat $\equiv_c$ die Darstellung:

$$\equiv_c \;=\; (\to_c^* \circ {}^*_c\!\leftarrow)\,.$$

Nun schätzen wir die Reduktion von $\hat{\mathcal{K}}_c^r$ ab:

$$\begin{array}{lll} (\equiv_c \circ \to_r \circ \equiv_c) & \subseteq & (\to_c^* \circ {}^*_c\!\leftarrow \circ \to_r \circ \equiv_c) \qquad \text{Darstellung von } \equiv_c \\ & \subseteq & (\to_c^* \circ \to_r \circ {}^*_c\!\leftarrow \circ \equiv_c) \qquad (\textit{Dec3}) \end{array}$$

Im nächsten Schritt wenden wir die Definition und anschließend die Darstellung von $\equiv_c$ an und erhalten:

$$({}^*_c\!\leftarrow \circ \equiv_c) \;\subseteq\; \equiv_c \;\subseteq\; (\to_c^* \circ {}^*_c\!\leftarrow)\,.$$

Durch Kombination der voranstehenden Abschätzungen folgt für die Reduktion von $\hat{\mathcal{K}}_c^r$:

$$(\equiv_c \circ \to_r \circ \equiv_c) \;\subseteq\; (\to_c^* \circ \to_r \circ {}^*_c\!\leftarrow \circ \equiv_c) \;\subseteq\; ({}^c\!\to_r \circ {}^*_c\!\leftarrow)\,,$$

und somit gilt die behauptete Darstellung von $\hat{\mathcal{K}}_c^r$. Die uniforme Konfluenz von $\hat{\mathcal{K}}_c^r$ folgt unmittelbar aus der uniformen Konfluenz von $\to_r$ und den Vertauschungseigenschaften der Kontrolle:

$$\begin{array}{lll} ((\equiv_c \circ {}_r\!\leftarrow \circ \equiv_c) \circ (\equiv_c \circ \to_r \circ \equiv_c)) & & \\ \quad \subseteq\; (\equiv_c \circ {}_r\!\leftarrow \circ \to_c^* \circ {}^*_c\!\leftarrow \circ \to_r \circ \equiv_c) & & \text{Darstellung von } \equiv_c \\ \quad \subseteq\; (\equiv_c \circ \to_c^* \circ {}_r\!\leftarrow \circ \to_r \circ {}^*_c\!\leftarrow \circ \equiv_c) & & (\textit{Dec2}) \\ \quad \subseteq\; (\equiv_c \circ {}_r\!\leftarrow \circ \to_r \circ \equiv_c) & & \text{Definition } \equiv_c \\ \quad \subseteq\; (\equiv_c \circ \to_r \circ {}_r\!\leftarrow \circ \equiv_c) \;\cup\; (\equiv_c \circ \equiv \circ \equiv_c) & & (\textit{Dec1}) \\ \quad \subseteq\; ((\equiv_c \circ \to_r \circ \equiv_c) \circ (\equiv_c \circ {}_r\!\leftarrow \circ \equiv_c)) \cup \equiv_c & & \square \end{array}$$

Lemma 2.6.4 *Ist $(\kappa_i)_i$ eine Berechnung von κ in $\mathcal{K}(\zeta)$, dann existiert eine Berechnung $(\kappa'_i)_i$ von κ in $\hat{\mathcal{K}}_c^r$ mit $l^r_{\mathcal{K}(\zeta)}((\kappa_i)_i) = l_{\hat{\mathcal{K}}_c^r}((\kappa'_i)_i)$.*

Beweis. Für endliche $(\kappa_i)_i$ folgt die Behauptung mit Induktion über $n = l^r_{\mathcal{K}(\zeta)}((\kappa_i)_i)$. Zu beachten ist, daß wir die Maximalität der zu konstruierenden Ableitung $(\kappa'_i)_i$ sicherstellen müssen. Aus diesem Grunde benötigen wir für $n = 0$ die Darstellung von Lemma 2.6.3. Desweiteren müssen wir für $n > 0$ die Induktionsvoraussetzung stets am Ende von $(\kappa_i)_i$ ansetzen.

Ist $(\kappa_i)_i$ unendlich, so folgt aus der Terminierung von $\to_c$, daß $(\kappa_i)_i$ unendlich viele $\to_r$ Schritte enthält. Dies erlaubt, für alle n eine Ableitung $(\kappa'_i)_{i=0}^n$ mit Induktion über n zu konstruieren. $\square$

Lemma 2.6.5 *Jede Berechnung in $\mathcal{K}_c^r$ ist auch eine Berechnung in $\hat{\mathcal{K}}_c^r$.*

Beweis. Offensichtlich ist jede Ableitung in $\mathcal{K}_c^r$ auch eine Ableitung in $\hat{\mathcal{K}}_c^r$. Wegen der Darstellung von $\hat{\mathcal{K}}_c^r$ in Lemma 2.6.3 fällt Irreduzibilität in beiden Kalkülen zusammen, und somit auch die Maximalität von Ableitungen. □

Beweis von Satz 2.6.2. Nach Lemma 2.6.3 ist $\hat{\mathcal{K}}_c^r$ uniform konfluent. Sei $\kappa \in K$ und $(\kappa_i)_i$ eine beliebige Berechnung von κ in $\mathcal{K}(\zeta)$. Wegen Lemma 2.6.4 existiert eine Berechnung $(\kappa_i')_i$ von κ in $\hat{\mathcal{K}}_c^r$ mit

$$l^r_{\mathcal{K}(\zeta)}((\kappa_i)_i) \;=\; l_{\hat{\mathcal{K}}_c^r}((\kappa_i')_i)\,.$$

Wegen Uniformität ist die rechte Seite unabhängig von der speziellen Berechnung $(\kappa_i')_i$ und somit ist auch die linke Seite unabhängig von $(\kappa_i)_i$, also $l^r_{\mathcal{K}(\zeta)}$ uniform.

Ist $(\kappa_i'')_i$ eine Berechnung von κ in $\mathcal{K}_c^r$, dann ist $(\kappa_i'')_i$ nach Lemma 2.6.5 auch eine Berechnung von κ in $l_{\hat{\mathcal{K}}_c^r}$ und

$$l_{\mathcal{K}_c^r}((\kappa_i'')_i) \;=\; l_{\hat{\mathcal{K}}_c^r}((\kappa_i'')_i)\,.$$

Daraus folgt die Uniformität von $l_{\mathcal{K}_c^r}$ und $l^r_{\mathcal{K}(\zeta)}(\kappa) = l_{\hat{\mathcal{K}}_c^r}(\kappa) = l_{\mathcal{K}_c^r}(\kappa)$. □

2.7 Adäquate Einbettungen

Ein wichtiges Ergebnis dieser Arbeit ist der Beweis, daß die in der Einleitung suggerierte Einbettung des strikten λ-Kalküls in den δ-Kalkül adäquat ist. Das dazu benötigte Beweisprinzip arbeiten wir nun heraus (Satz 2.7.4). Anschließend bereiten wir weitere Adäquatheitsbeweise vor.

Seien $\mathcal{K} = (K, \equiv_K, \to_K)$ und $\mathcal{L} = (L, \equiv_L, \to_L)$ zwei Kalküle. Eine Einbettung von $\mathcal{K}$ nach $\mathcal{L}$ ist eine Funktion $\Phi : K \to L$, die linksinvariant unter $\equiv_K$ und rechtsinvariant unter $\equiv_L$ ist. Seien γ_K und γ_L uniforme Komplexitätsmaße für $\mathcal{K}$ bzw $\mathcal{L}$. Dann heißt Φ adäquat bezüglich γ_K und γ_L, falls

$$\gamma_K(\kappa) \;=\; \gamma_L(\Phi(\kappa))$$

für alle κ gilt. Φ heißt adäquat bezüglich Terminierung, wenn Φ adäquat bezüglich der Terminierungsmaße $t_{\mathcal{K}}$ und $t_{\mathcal{L}}$ ist. In diesem Fall terminiert κ genau dann in $\mathcal{K}$, wenn $\Phi(\kappa)$ in $\mathcal{L}$ terminiert.

Definition 2.7.1 *Wir nennen eine Teilmenge $S \subseteq K \times L$* Längensimulation *zwischen $\mathcal{K}$ und $\mathcal{L}$, falls S die folgenden beiden Eigenschaften erfüllt:*

(LS1) Falls $(\kappa, \lambda) \in S$ *und* κ *irreduzibel in* $\rightarrow_K$*, dann ist* λ *irreduzibel in* $\rightarrow_L$*.*

(LS2) Falls $(\kappa, \lambda) \in S$ *und* $\kappa \rightarrow_K \kappa'$*, dann existiert* λ' *mit* $\lambda \rightarrow_L \lambda'$ *und* $(\kappa', \lambda') \in S$*.*

Ist weiterhin $\Phi : K \rightarrow L$ *eine Einbettung von* $\mathcal{K}$ *nach* $\mathcal{L}$*, dann nennen wir* S Längensimulation für Φ*, falls* S *alle Paare der Form* $(\kappa, \Phi(\kappa))$ *enthält.*

Lemma 2.7.2 *Sei* S *eine Längensimulation zwischen Kalkülen* $\mathcal{K}$ *und* $\mathcal{L}$*. Dann existiert für alle Paare* $(\kappa, \lambda) \in S$ *und alle Berechnungen von* κ *eine Berechnung von* λ *der gleichen Länge.*

Beweis. Einfach. Wir müssen die Fälle für endliche und unendliche Berechnungen unterscheiden, und darauf achten, daß die zu konstruierenden Ableitungen maximal sind. □

Lemma 2.7.3 *Sei* $\mathcal{L}$ *uniform,* Φ *eine Einbettung von* $\mathcal{K}$ *nach* $\mathcal{L}$*,* S *eine Längensimulation für* Φ *und* κ *ein Ausdruck von* $\mathcal{K}$*. Dann gibt es zu jeder Berechnung von* κ *eine Berechnung* $\Phi(\kappa)$ *der gleichen Länge.*

Beweis. Per Definition gilt $(\kappa, \Phi(\kappa)) \in S$ für alle κ. Somit folgt die Behauptung unmittelbar aus Lemma 2.7.2. □

Umgekehrt gibt es aber nicht zu jeder Berechnung von $\Phi(\kappa)$ eine Berechnung von κ. Dies ist jedoch nicht tragisch, da sich alle Berechnungen von $\Phi(\kappa)$ wegen Uniformität gleich verhalten. In diesem Punkte unterscheiden sich die Methoden dieser Arbeit von denjenigen von Robin Milner [Mil92]. Denn um mit der Nicht-Uniformität des π-Kalküls zurecht zu kommen, stellt Robin Milner eine Bijektion zwischen Berechnungen sicher. Dazu verwendet er *Längenbisimulationen*, also Längensimulationen, deren Inverse auch Längensimulationen sind. Diese Methode ist für unsere Zwecke nicht hinreichend.

Satz 2.7.4 *Sei* $\mathcal{L}$ *uniform und* Φ *eine Einbettung von* $\mathcal{K}$ *nach* $\mathcal{L}$*, für die eine Längensimulation existiert. Dann ist* $\mathcal{K}$ *uniform und* Φ *adäquat bezüglich der Längenmaße.*

Beweis. Sei κ ein Ausdruck von $\mathcal{K}$ und $(\kappa_i)_i$ und $(\kappa'_i)_i$ Berechnungen von κ. Dann existieren nach Lemma 2.7.3 Berechnungen $(\lambda_i)_i$ und $(\lambda'_i)_i$ von $\Phi(\kappa)$ mit

$$l_{\mathcal{K}}((\kappa_i)_i) \;=\; l_{\mathcal{L}}((\lambda_i)_i) \qquad \text{und} \qquad l_{\mathcal{K}}((\kappa'_i)_i) \;=\; l_{\mathcal{L}}((\lambda'_i)_i)\,.$$

Aus der Uniformität von $\mathcal{L}$ folgt $l_{\mathcal{L}}((\lambda_i)_i) = l_{\mathcal{L}}((\lambda'_i)_i)$ und somit gilt auch $l_{\mathcal{K}}((\kappa_i)_i) = l_{\mathcal{K}}((\kappa'_i)_i)$. Da $(\kappa_i)_i$ und $(\kappa'_i)_i$ beliebig waren, haben also alle Berechnungen von κ die gleiche Länge. Weiterhin folgt Adäquatheit:

$$l_{\mathcal{K}}(\kappa) \;=\; l_{\mathcal{K}}((\kappa_i)_i) \;=\; l_{\mathcal{L}}((\lambda_i)_i) \;=\; l_{\mathcal{L}}(\Phi(\kappa))\,.$$ □

Nun formulieren wir ein sehr nützliches Kriterium, daß die Gleichwertigkeit verschiedener Zerlegungen sicherstellt. Dadurch lassen sich zwei Kalküle als gleichwertig nachweisen, die die gleichen Ausdrücke und Kongruenzen, aber leicht unterschiedliche Reduktionen besitzen.

Genauer betrachtet interessiert uns die folgende Situation: Gegeben ist ein Kalkül, dessen Reduktion $(\rightarrow_r \cup \rightarrow_c)$ sich in eine eigentliche Reduktion $\rightarrow_r$ und eine Kontrolle $\rightarrow_c$ zerlegen läßt. Dann stellt sich die Frage, unter welchen Bedingungen wir $\rightarrow_c$ durch eine stärkere Kontrolle $\rightarrow_d$ ersetzen können, ohne die eigentliche Reduktion $\rightarrow_r$ zu beeinflussen.

Satz 2.7.5 *Seien* $\zeta_c = (\rightarrow_r, \rightarrow_c)$ *und* $\zeta_d = (\rightarrow_r, \rightarrow_d)$ *zwei Zerlegungen für* $(K, \equiv)$ *mit*

1. *Schritte in* $\rightarrow_d$ *erhalten Irreduzibilität bezüglich* $(\rightarrow_r \cup \rightarrow_c)$.
2. $\rightarrow_c \subseteq (\rightarrow_d \cup {}_d\!\leftarrow)^*$.

Dann gilt $l^r_{\mathcal{K}(\zeta_c)} = l^r_{\mathcal{K}(\zeta_d)}$ *und somit ist die Identität eine adäquate Einbettung von* $\mathcal{K}(\zeta_c)$ *nach* $\mathcal{K}(\zeta_d)$ *bezüglich der r-Längenmaße.*

Zum Beweis benötigen wir ein Lemma für das wir die Voraussetzungen von Satz 2.7.5 annehmen. Es formuliert zwei Vertauschungseigenschaften, in denen zusätzlich $\rightarrow_c$ durch $\rightarrow_d$ ersetzt wird. Wir erinnern an die Definitionen $\overset{c}{\rightarrow}_r = (\rightarrow_c^* \circ \rightarrow_r \circ \rightarrow_c^*)$ bzw. $\overset{d}{\rightarrow}_r = (\rightarrow_d^* \circ \rightarrow_r \circ \rightarrow_d^*)$.

Lemma 2.7.6 1. $({}^*_d\!\leftarrow \circ \rightarrow_c^*) \subseteq (\rightarrow_d^* \circ {}^*_d\!\leftarrow)$.
2. $({}^*_d\!\leftarrow \circ \overset{c}{\rightarrow}_r) \subseteq (\overset{d}{\rightarrow}_r \circ {}^*_d\!\leftarrow)$.

Beweis. Aus Voraussetzung 2 von Satz 2.7.5 folgt $({}^*_d\!\leftarrow \circ \rightarrow_c^*) \subseteq (\rightarrow_d \cup {}_d\!\leftarrow)^*$. Nach (*Dec2*) ist $\rightarrow_d$ konfluent und somit Church-Rosser, d.h. $(\rightarrow_d \cup {}_d\!\leftarrow)^* \subseteq (\rightarrow_d^* \circ {}^*_d\!\leftarrow)$, woraus Inklusion 1 folgt. Inklusion 2 können wir nun folgendermaßen nachweisen:

$$
\begin{array}{lll}
({}^*_d\!\leftarrow \circ \overset{c}{\rightarrow}_r) & \subseteq & ({}^*_d\!\leftarrow \circ \rightarrow_c^* \circ \rightarrow_r \circ \rightarrow_c^*) \qquad \text{Definition von } \overset{c}{\rightarrow}_r \\
& \subseteq & (\rightarrow_d^* \circ {}^*_d\!\leftarrow \circ \rightarrow_r \circ \rightarrow_c^*) \qquad \text{Inklusion 1} \\
& \subseteq & (\rightarrow_d^* \circ \rightarrow_r \circ {}^*_d\!\leftarrow \circ \rightarrow_c^*) \qquad (\textit{Dec3}) \\
& \subseteq & (\rightarrow_d^* \circ \rightarrow_r \circ \rightarrow_d^* \circ {}^*_d\!\leftarrow) \qquad \text{Inklusion 1} \\
& \subseteq & (\overset{d}{\rightarrow}_r \circ {}^*_d\!\leftarrow) \qquad \text{Definition von } \overset{d}{\rightarrow}_r
\end{array}
$$

□

Beweis von Satz 2.7.5. Wir betrachten die Kalküle $\mathcal{K}_c^r = (K, \equiv, \overset{c}{\to}_r)$ und $\mathcal{K}_d^r = (K, \equiv, \overset{d}{\to}_r)$. Nach Satz 2.6.2 ist die Behauptung des Satzes äquivalent zur Adäquatheit von $Id : \mathcal{K}_c^r \to \mathcal{K}_d^r$ bezüglich der Längenmaße. Um dies nachzuweisen, wenden wir Satz 2.7.4 mit folgender Simulation S an:

$$(\kappa, \kappa') \in S \qquad gdw. \qquad \kappa \to_d^* \kappa'.$$

Wir müssen zeigen, daß S eine Längensimulation zwischen $\mathcal{K}_c^r$ und $\mathcal{K}_d^r$ ist. Denn die Uniformität von $\mathcal{K}_d^r$ folgt wiederum aus Satz 2.6.2, und außerdem enthält S alle Paare der Form (κ, κ), ist also eine Simulation für die Identität.

Wir zeigen nun Eigenschaft (*LS1*). Dazu setzen wir κ_1 als irreduzibel in $\overset{c}{\to}_r$ und $(\kappa_1, \kappa_2) \in S$ voraus, und zeigen die Irreduzibilität von κ_2 in $\overset{d}{\to}_r$. Da $\to_c$ terminiert, existiert κ_1' mit $\kappa_1 \to_c^* \kappa_1'$ und κ_1' irreduzibel in $\to_c$. Da κ_1 irreduzibel in $\overset{c}{\to}_r$, ist κ_1' irreduzibel in $(\to_r \cup \to_c)$. Aus Teil 1 von Lemma 2.7.6 folgt die Existenz von κ_2' mit $\kappa_2 \to_d^* \kappa_2'$ und $\kappa_1' \to_d^* \kappa_2'$. Da $\to_d$ terminiert, existiert κ_2'' irreduzibel in $\to_d$ mit $\kappa_2' \to_d^* \kappa_2''$.

Nach Voraussetzung 2 von Proposition 2.7.5 ist auch κ_2'' irreduzibel in $(\to_r \cup \to_c)$, also irreduzibel in $(\to_r \cup \to_d)$ und auch in $\overset{d}{\to}_r$. Wegen (*Dec2*) und (*Dec3*) gilt:

$$({}_d\overset{r}{\leftarrow} \circ \to_d^*) \subseteq (\to_d^* \circ {}_d\overset{r}{\leftarrow}) .$$

Wäre nun κ_2 reduzierbar in $\overset{d}{\to}_r$, dann auch κ_2'', was wir bereits ausgeschlossen haben. Also ist κ_2 irreduzibel in $\overset{d}{\to}_r$, wie zu zeigen war.

Wir weisen nun Eigenschaft (*LS2*) nach. Dazu seien $\kappa_1, \kappa_1', \kappa_2$ mit $\kappa_1 \overset{c}{\to}_r \kappa_1'$ und $(\kappa_1, \kappa_2) \in S$ gegeben. Die Existenz von κ_2' mit $\kappa_2 \overset{d}{\to}_r \kappa_2'$ und $(\kappa_1', \kappa_2') \in S$ folgt unmittelbar aus Lemma 2.7.6 Teil 2. □

In Analogie zu Satz 2.7.4 können wir Adäquatheit bezüglich Terminierung nachweisen, selbst wenn Adäquatheit bezüglich Komplexität nicht gilt. Ein mögliches Kriterium erhalten wir mit folgender Definition.

Definition 2.7.7 *Wir nennen eine Teilmenge $S \subseteq K \times L$* Terminierungssimulation *zwischen $\mathcal{K}$ und $\mathcal{L}$, falls S die folgenden beiden Eigenschaften erfüllt:*

1. *Falls $(\kappa, \lambda) \in S$ und κ irreduzibel in $\to_K$, dann ist λ irreduzibel in $\to_L$.*
2. *Falls $(\kappa, \lambda) \in S$ und $\kappa \to_K \kappa'$, dann existiert λ' mit $\lambda \to_L^* \lambda'$ und $(\kappa', \lambda') \in S$.*

Eine Terminierungssimulation für eine Einbettung Φ *von $\mathcal{K}$ nach $\mathcal{L}$ ist eine Terminierungssimulation zwischen $\mathcal{K}$ und $\mathcal{L}$, die alle Paare der Form $(\kappa, \Phi(\kappa))$ enthält.*

Die Definition ist vollkommen identisch zu derjenigen von Längensimulation, bis auf die Verwendung von $\rightarrow^*$ in der zweiten Eigenschaft.

Satz 2.7.8 *Sei Φ eine Einbettung von $\mathcal{K}$ in einen Kalkül $\mathcal{L}$, dessen Terminierungsmaß uniform ist. Wenn eine Terminierungssimulation für Φ existiert, dann ist das Terminierungsmaß für C_K uniform und Φ adäquat bezüglich Terminierung.*

Beweis. Analog zum Beweis von Satz 2.7.4 für Längensimulationen. □

2.8 Basen

Für die Analyse von Kalkülen ist es häufig wünschenswert, Eigenschaften aller Ausdrücke lediglich auf standardisierten Ausdrücken nachzuweisen. So können wir zum Beispiel den Beweis der uniformen Konfluenz des δ-Kalküls auf die Betrachtung von α-standardisierten Pränexnormalformen zurückführen. Das Konzept einer Reduktionsbasis eines Kalküls extrahiert die dazu notwendigen Bedingungen.

Ein ähnliches Problem tritt auf, wenn man Relationen oder Funktionen auf Ausdrücken definieren möchte, die invariant unter Kongruenz sind. Dabei kann man zum Beispiel an eine Terminierungsordnung denken. In solchen Fällen ist das Konzept einer Definitionsbasis eines Kalküls hilfreich.

Seien $(K, \equiv_K)$, $(L, \equiv_L)$ und $(M, \equiv_M)$ Mengen mit Äquivalenzrelationen. K liegt dicht in L bezüglich $\equiv_L$, falls $K \subseteq L$ und falls für alle λ ein κ existiert mit $\lambda \equiv_L \kappa$.

Definition 2.8.1 *Ein Paar $(K, \equiv_K)$ heißt* Definitionsbasis *von $(L, \equiv_L)$, falls gilt:*

1. *K liegt dicht in L bezüglich $\equiv_L$.*
2. *Die Einschränkung von $\equiv_L$ auf $K \times K$ stimmt mit $\equiv_K$ überein.*

Eine Funktion ξ auf K heißt invariant bezüglich $\equiv_K$, falls ξ als Relation aufgefaßt linksinvariant unter $\equiv_K$ ist, also wenn $\xi(\kappa_1) = \xi(\kappa_2)$ für alle κ_1, κ_2 mit $\kappa_1 \equiv_K \kappa_2$.

Proposition 2.8.2 *Sei $(K, \equiv_K)$ eine Definitionsbasis von $(L, \equiv_L)$ und ξ eine Funktion auf K, die invariant unter $\equiv_K$ ist. Dann läßt sich ξ eindeutig zu einer Funktion auf L fortsetzen, die invariant unter $\equiv_L$ ist.*

Beweis. Wir können ξ_1 wie folgt auf beliebigen λ definieren:

$$\xi_1(\lambda) = \xi(\kappa) \quad \text{falls } \lambda \equiv_L \kappa \text{ und } \kappa \in K.$$

ξ_1 ist total, denn wegen Dichtheit von K in L existiert zu jedem λ ein κ mit $\lambda \equiv_L \kappa$. ξ_1 ist eine Funktion, denn für alle λ ist $\xi_1(\lambda)$ eindeutig bestimmt. Um dies zu zeigen, seien $\kappa_1, \kappa_2 \in K$ mit $\lambda \equiv_L \kappa_1$ und $\lambda \equiv_L \kappa_2$. Da $\equiv_L$ eine Äquivalenzrelation ist, gilt $\kappa_1 \equiv_L \kappa_2$. Aus der zweiten Eigenschaft einer Definitionsbasis folgt $\kappa_1 \equiv_K \kappa_2$. Da ξ invariant bezüglich $\equiv_K$ ist, erhalten wir $\xi_1(\kappa_1) = \xi_1(\kappa_2)$.

Die Funktion ξ_1 ist per Definition eine Fortsetzung von ξ, die invariant unter $\equiv_L$ ist. Als solche ist sie eindeutig bestimmt, denn ist ξ_2 eine weitere solche Fortsetzung und $\lambda \in L$, dann existiert $\kappa \in K$ mit $\lambda \equiv_L \kappa$ und somit: $\xi_1(\lambda) = \xi_1(\kappa) = \xi(\kappa) = \xi_2(\kappa) = \xi_2(\lambda)$. □

Definitionsbasen können auch zur Definition von Relationen verwendet werden, die invariant unter Kongruenz sind.

Proposition 2.8.3 *Sei $(K, \equiv_K)$ eine Definitionsbasis von $(L, \equiv_L)$. Dann läßt sich jede Relation $\rightarrow$ auf $K \times M$, die linksinvariant unter $\equiv_K$ ist, eindeutig zu einer Relation auf $L \times M$ fortsetzen, die linksinvariant unter $\equiv_L$ ist.*

Beweis. Wir definieren eine Relation $\rightarrow_1$ auf $L \times M$ durch

$$\rightarrow_1 \; = \; (\equiv_{L|(L\times K)} \circ \rightarrow) \, .$$

Wegen der Transitivität von $\equiv_L$ ist $\rightarrow_1$ linksinvariant unter $\equiv_L$. Wir zeigen, daß $\rightarrow_1$ eine Fortsetzung von $\rightarrow$ ist. Per Definition gilt:

$$\rightarrow_{1|(K\times M)} \; = \; (\equiv_{L|(L\times K)} \circ \rightarrow)_{|(K\times M)} \; = \; (\equiv_{L|(K\times K)} \circ \rightarrow)$$

Aus der zweiten Eigenschaft einer Definitionsbasis und der Linksinvarianz von $\rightarrow$ unter $\equiv_K$ folgt:

$$(\equiv_{L|(K\times K)} \circ \rightarrow) \; = \; (\equiv_K \circ \rightarrow) \; \subseteq \; \rightarrow \; .$$

In der letzten Abschätzung gilt sogar Gleichheit wegen der Reflexivität von $\equiv_K$ und somit erhalten wir

$$\rightarrow_{1|(K\times M)} \; = \rightarrow \; .$$

Nun beweisen wir die Eindeutigkeit der Fortsetzung von $\rightarrow$, die linksinvariant unter $\equiv_L$ ist. Sei dazu $\rightarrow_2$ eine zweite solche Fortsetzung. Wir nehmen $\lambda \rightarrow_1 \mu$ an und zeigen, daß daraus $\lambda \rightarrow_2 \mu$ folgt. Dies ist aus Symmetriegründen hinreichend für Eindeutigkeit. Da K dicht in L bezüglich $\equiv_L$ liegt, existiert ein κ mit $\kappa \equiv_L \lambda$. Wegen der Linksinvarianz von $\rightarrow_1$ unter $\equiv_L$ folgt $\kappa \rightarrow_1 \mu$. Da $\rightarrow_1$ und $\rightarrow_2$ Fortsetzungen von $\rightarrow$ sind, gilt $\kappa \rightarrow \mu$ und $\kappa \rightarrow_2 \mu$. Aus der Linksinvarianz von $\rightarrow_2$ unter $\equiv_L$ folgt $\lambda \rightarrow_2 \mu$. □

Proposition 2.8.4 *Sei $\rightarrow$ eine Relation auf $L \times M$, die linksinvariant unter $\equiv_L$ ist. Liegt K dicht in L bezüglich $\equiv_L$ und ist die Einschränkung von $\rightarrow$ auf $K \times M$ rechtsinvariant unter $\equiv_M$, dann ist $\rightarrow$ auch rechtsinvariant unter $\equiv_M$.*

Beweis. Wir nehmen $\lambda \rightarrow \mu_1 \equiv_M \mu_2$ an und zeigen $\lambda \rightarrow \mu_2$. Wegen Dichtheit existiert κ mit $\kappa \equiv_L \lambda$ und somit folgt aus Linksinvarianz $\kappa \rightarrow \mu_1$. Wegen Rechtsinvarianz von $\rightarrow_{|(K \times M)}$ gilt $\kappa \rightarrow \mu_2$ und wiederum wegen Linksinvarianz folgt $\lambda \rightarrow \mu_2$. □

Definition 2.8.5 *Sei $\overset{-}{\rightarrow}$ eine Relation auf $K \times L$. Dann heißt $(K, \equiv_K, \overset{-}{\rightarrow})$* Reduktionsbasis *eines Kalküls $(L, \equiv_L, \rightarrow)$, falls gilt:*

1. *K liegt dicht in L bezüglich $\equiv_L$.*
2. *Für alle $\kappa \in K$ und $\lambda \in L$ mit $\kappa \rightarrow \lambda$ gilt $\kappa\ (\equiv_K \circ \overset{-}{\rightarrow} \circ \equiv_L)\ \lambda$.*

Üblicherweise werden Reduktionsbasen zusätzlich $\equiv_K \subseteq \equiv$ und $\overset{-}{\rightarrow} \subseteq \rightarrow$ erfüllen. Dies ist für uns nicht wesentlich, würde aber die Umkehrungen der folgenden Propositionen implizieren.

Proposition 2.8.6 *Sei $(K, \equiv_K, \overset{-}{\rightarrow})$ eine Reduktionsbasis von $\mathcal{L} = (L, \equiv_L, \rightarrow)$. Dann ist $\mathcal{L}$ uniform konfluent, wenn $(\overset{-}{\leftarrow} \circ \equiv_K \circ \overset{-}{\rightarrow}) \subseteq ((\rightarrow \circ \leftarrow) \cup \equiv_L)$.*

Beweis. Wir nehmen an, daß $\lambda_1 \leftarrow \lambda \rightarrow \lambda_2$ gilt. Die Dichtheit von K in L impliziert die Existenz von κ mit $\kappa \equiv_L \lambda$. Für dieses κ gilt $\lambda_1 \leftarrow \kappa \rightarrow \lambda_2$ wegen der Kalküleigenschaft für $\mathcal{L}$. Aus der Definition einer Reduktionsbasis folgt:

$$\lambda_1\ (\equiv_L \circ \overset{-}{\leftarrow} \circ \equiv_K \circ \overset{-}{\rightarrow} \circ \equiv_L)\ \lambda_2\ .$$

Die Bedingung der Proposition impliziert nun $\lambda_1 \equiv_L \lambda_2$ oder $\lambda_1\ (\rightarrow \circ \leftarrow)\ \lambda_2$, wozu wir wiederum die Kalküleigenschaft ausnutzen. □

Proposition 2.8.7 *Sei $(K, \equiv_K, \overset{-}{\rightarrow})$ eine Reduktionsbasis von $\mathcal{L} = (L, \equiv_L, \rightarrow)$. Dann ist $\mathcal{L}$ stark konfluent, wenn $(\overset{-}{\leftarrow} \circ \equiv_K \circ \overset{-}{\rightarrow}) \subseteq ((\rightarrow^{\epsilon} \circ {}^{\epsilon}\leftarrow)$.*

Beweis. Analog zu Proposition 2.8.6. □

Proposition 2.8.8 *Sei $(K, \equiv_K, \overset{-}{\rightarrow}_1)$ eine Reduktionsbasis von $(L, \equiv_L, \rightarrow_1)$ und $(K, \equiv_K, \overset{-}{\rightarrow}_2)$ eine Reduktionsbasis von $(L, \equiv_L, \rightarrow_2)$. Dann vertauschen $\rightarrow_1$ und $\rightarrow_2$, falls $({}_1\overset{-}{\leftarrow} \circ \equiv_K \circ \overset{-}{\rightarrow}_2) \subseteq (\rightarrow_2 \circ {}_1\leftarrow)$.*

Beweis. Analog zu Proposition 2.8.6. □

Eine analoge Aussage gilt natürlich auch für lokale Konfluenz. Da wir diese allerdings nicht benötigen, formulieren wir sie nicht explizit.

Proposition 2.8.9 *Sei* $(K, \equiv_K, \bar{\rightarrow})$ *eine Reduktionsbasis von* $\mathcal{L} = (L, \equiv_L, \rightarrow)$. *Dann ist* λ *irreduzibel in* $\mathcal{L}$, *falls ein* κ *existiert mit* $\kappa \equiv_L \lambda$ *und* κ *irreduzibel in* $(\equiv_K \circ \bar{\rightarrow})$ *ist.*

Beweis. Offensichtlich. □

Trotz seiner Einfachheit wird uns dieses Kriterium erlauben, die Irreduzibilität von Ausdrücken in allen betrachteten Kalkülen zu entscheiden.

Kapitel 3

Uniform nebenläufige Berechnung

Wir stellen den δ-Kalkül und eine Variante des δ-Kalküls mit Anullierung vor und untersuchen beide im Hinblick auf Konfluenz und Uniformität. Auf technischer Seite geben wir Definitions- und Reduktionsbasen an und beweisen alle behaupteten Konfluenz und Uniformitätseigenschaften. Ferner weisen wir die Gleichwertigkeit des δ-Kalküls und seiner Variante im Hinblick auf Terminierung und Komplexität nach.

3.1 Der δ-Kalkül

Wir setzen einen unendlichen Vorrat an Variablen voraus. Diese denotieren wir mit x, y, z, u, v, w, r, s und t. Mit $\overline{x}$ bezeichnen wir eine endliche Folge von Variablen und mit $\mathcal{V}(\overline{x})$ die Menge der in $\overline{x}$ enthaltenen Variablen. Für die Komposition zweier Folgen $\overline{x}$ und $\overline{y}$ schreiben wir $\overline{x}\,\overline{y}$. Desweiteren fassen wir Variablen frei als einelementige Folgen auf, was zum Beispiel $x\,\overline{y}$ definiert. Wir nennen eine Folge linear, falls ihre Komponenten paarweise verschieden sind. Später werden wir analoge Notationen für Folgen über anderen syntaktischen Kategorien benutzen.

Mit Δ bezeichnen wir die Menge aller Ausdrücken E, F und G, die wir in Abbildung 3.1 definieren.

In voller Schönheit sind δ-Ausdrücke vollständig geklammert. Wir schreiben diese Klammern aber nur optional und vereinbaren, daß Abstraktion und Deklaration höhere Priorität als Komposition besitzen. Zum Beispiel gelten $x{:}\overline{y}/E \wedge F = (x{:}\overline{y}/E) \wedge F$ und $\exists x\,E \wedge F = (\exists x\,E) \wedge F$.

Eine benannte Abstraktion $x{:}\overline{y}/E$ besteht aus einer Variablen x und einer Ab-

E, F, G	$::=$	$x{:}\overline{y}/E$	*Abstraktion*
		$x\,\overline{y}$	*Applikation*
		$x = y$	*Gleichung*
		$E \wedge F$	*Komposition*
		$\exists x\,E$	*Deklaration*
		$\top$	*Null*[1]

Abbildung 3.1: Ausdrücke

straktion $\overline{y}/\,E$, wobei wir x als Referenz auf $\overline{y}/\,E$ interpretieren. Die Variablen in $\overline{y}$ nennen wir formale Argumente und E Rumpf der Abstraktion. Der Rumpf wird bezüglich der formalen Argumente abstrahiert. Die Metavariable A steht im folgenden stets für eine Abstraktion $\overline{y}/\,E$. Zum Beispiel kürzen wir $x{:}\overline{y}/E$ mit $x{:}\,A$ ab, wobei A für $\overline{y}/\,E$ steht.

Die Variable x einer Applikation $x\,\overline{y}$ fungiert als Referenz auf eine zu applizierende Abstraktion, die Variablen $\overline{y}$ dienen als aktuelle Argumente. Für eine Folge von Deklarationen $\exists y_1 \ldots \exists y_n\,E$ schreiben wir abkürzend $\exists \overline{y}\,E$, wobei $\overline{y}$ für $y_1 \ldots y_n$ steht. Desweiteren benötigen wir eine feinkörnige Schreibweise für simultanes Abstrahieren in Analogie zum Deklarieren. Dazu schreiben wir $\overline{u}A$ abkürzend für $(\overline{u}\overline{y})/\,E$, falls A für $\overline{y}/\,E$ steht.

Die Variablenbinder des δ-Kalküls sind Abstraktionen und Deklarationen. Ausdrücke der Form xA und $\exists x\,E$ sind Variablenbinder für x mit Skopus A bzw. E. Eine Variable x heißt gebunden in E, falls ein Variablenbinder für x in E existiert. Eine Variable x heißt frei in E, falls es ein Auftreten von x in E gibt, das nicht im Skopus eines Variablenbinders für x liegt. Mit $\mathcal{BV}(E)$ bezeichnen wir die Menge der in E gebundenen Variablen und mit $\mathcal{FV}(E)$ die Menge der in E freien Variablen. Die Menge aller Variablen in E denotieren wir mit $\mathcal{V}(E)$, so daß $\mathcal{V}(E) = \mathcal{FV}(E) \cup \mathcal{BV}(E)$ gilt.

Eine Kongruenz $\equiv$ auf Δ ist eine Äquivalenzrelation auf Δ, die in beliebigen Kontexten angewendet werden darf. Wir fordern also, daß jede Kongruenz die folgenden Regeln erfüllt:

$$\frac{E \equiv F}{\exists x\,E \equiv \exists x\,F} \qquad \frac{E \equiv F}{E \wedge G \equiv F \wedge G} \qquad \frac{E \equiv F}{G \wedge E \equiv G \wedge F} \qquad \frac{E \equiv F}{x{:}\overline{y}/E \equiv x{:}\overline{y}/F}$$

Die erste Regel ist zum Beispiel so zu lesen, daß für alle x, E und F, die der Voraussetzung $E \equiv F$ genügen, die Konklusion $\exists x\,E \equiv \exists x\,F$ gilt.

[1] Die Null $\top$ ist in dieser Formulierung des δ-Kalküls überflüssig. Wir werden sie allerdings beim Rechnen mit Pränexnormalformen in Abschnitt 3.4 benötigen. Auch im Zusammenhang mit Annullierung in Abschnitt 3.3 ist es von Vorteil, eine Identität bezüglich Komposition zur Verfügung zu haben.

Unter einer Substitution σ verstehen wir eine Abbildung von Variablen auf Variablen! Zum Beispiel steht der Operator $[y/x]$ für diejenige Substitution, die x auf y abbildet, und alle anderen Variablen auf sich selbst. Wir schreiben $\overline{y}\sigma$ für die elementweise Anwendung von σ auf $\overline{y}$ und $E\sigma$ für die *homomorphe* Anwendung einer Substitution σ auf E:

$$\begin{array}{rclcrcl}
(x = y)\,\sigma & = & \sigma(x) = \sigma(y)\,, & \qquad & \top\sigma & = & \top\,, \\
(x\colon A)\,\sigma & = & \sigma(x)\colon A\sigma\,, & & (\overline{y}/\,E)\,\sigma & = & \overline{y}\sigma/\,E\sigma\,, \\
(x\,\overline{y}\,)\,\sigma & = & \sigma(x)\,\overline{y}\sigma\,, & & (E \wedge F)\,\sigma & = & E\sigma \wedge F\sigma\,, \\
(\exists x\,E)\,\sigma & = & \exists\sigma(x)\,E\sigma\,. & & & &
\end{array}$$

Wir definieren die strukturelle Kongruenz $\equiv$ als kleinste Kongruenz auf Δ, die die Axiome und Regeln in Abbildung 3.2 erfüllt, wobei wir unter Axiomen Regeln ohne Voraussetzungen (und ohne Querstrich) verstehen.

(α)	$\exists x\,E \equiv \exists y\,E[y/x]$ falls $y \notin \mathcal{V}(E)$
	$\overline{u}xA \equiv \overline{u}yA[y/x]$ falls $y \notin \mathcal{V}(A)$
(ACI)	$(E \wedge F) \wedge G \equiv E \wedge (F \wedge G)$
	$E \wedge F \equiv F \wedge E$
	$E \wedge \top \equiv E$
$(Exch)$	$\exists x \exists y\,E \equiv \exists y \exists x\,E$
$(Scope)$	$\exists x\,E \wedge F \equiv \exists x(E \wedge F)$ falls $x \notin \mathcal{FV}(F)$

Abbildung 3.2: Strukturelle Kongruenz

Die strukturelle Kongruenz formalisiert die üblichen Eigenschaften von Variablenbindern und paralleler Komposition: Das Axiom (α) beschreibt die konsistente Umbenennung gebundener Variablen. Wegen (ACI) ist Komposition $\wedge$ bezüglich $\equiv$ assoziativ und kommutativ, und $\top$ die Identität bezüglich Komposition. Deklarationen können bezüglich $\equiv$ frei vertauscht werden $(Exch)$ ("Exchange") und der Skopus von deklarierten Variablen ist beweglich $(Scope)$.

Der simultane Substitutionsoperator $[\overline{y}/\overline{x}]$ verallgemeinert den Operator $[y/x]$. Er ist für alle gleichlangen Folgen $\overline{y}$ und $\overline{x}$ definiert, falls $\overline{x}$ linear ist. Gelten $\overline{y} = y_1 \ldots y_n$ und $\overline{x} = x_1 \ldots x_n$, dann bildet $[\overline{y}/\overline{x}]$ jeweils x_i auf y_i ab, und alle anderen Variablen auf sich selbst. Bei allen Anwendungen von $[\overline{y}/\overline{x}]$ setzen wir implizit seine Definiertheit voraus.

Die Reduktion $\rightarrow$ ist die kleinste binäre Relation auf Δ, die die Axiome und Regeln in Abbildung 3.3 erfüllt.

Applikation (A) führt Prozeduraufrufe aus, wobei eine Kopie des Prozedurrumpfes angelegt und formale durch aktuelle Argumente ersetzt werden. Elimination (E)

$$(A)\quad x{:}\overline{y}/E \wedge x\,\overline{z} \;\to\; x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}] \quad \text{falls } \mathcal{V}(\overline{z}) \cap \mathcal{BV}(E) = \emptyset$$

$$(E)\quad \exists y \exists x (x = y \wedge E) \;\to\; \exists y\, E[y/x] \quad \text{falls } x \neq y \text{ und } y \notin \mathcal{BV}(E)$$

$$(Comp)\quad \frac{E \to F}{E \wedge G \to F \wedge G} \qquad (Decl)\quad \frac{E \to F}{\exists x\, E \to \exists x\, F}$$

$$(Congr)\quad \frac{E_1 \equiv E_2 \qquad E_2 \to F_2 \qquad F_2 \equiv F_1}{E_1 \to F_1}$$

Abbildung 3.3: Reduktion

macht lokale Variablen gleich, indem sie Gleichungen zwischen lokalen Variablen elaboriert. Dadurch werden Referenzketten zwischen Abstraktion und Applikation verkürzt. Reduktion ist unterhalb von Deklaration (*Decl*) und Komposition (*Comp*) erlaubt, aber nicht in Rümpfen von Abstraktionen. Wegen (*Congr*) ist Reduktion invariant unter Kongruenz.

Es ist bemerkenswert, daß wir keine weiteren Mechanismen zur Simplifikation von Gleichungen benötigen. Insbesondere stört es nicht, daß wir triviale Gleichungen $x = x$ nicht entfernen können und daß Gleichungen mit globalen Variablen keinerlei Effekt haben. Wichtig ist jedoch, daß wir das folgende symmetrische Eliminationsaxiom ableiten können, obwohl wir Gleichungen bezüglich $\equiv$ nicht als symmetrisch vorausgesetzt haben:

$$\exists y \exists x (x = y \wedge E) \;\to\; \exists x\, E[x/y] \quad \text{falls } x \neq y \text{ und } x \notin \mathcal{BV}(E)\,.$$

Im Gegensatz zu (E) wird hier y und nicht etwa x eliminiert. Zur Ableitung von diesem Axiom verwenden wir (E), (α) und (*Congr*); wir setzen $x \neq y$ und $x \notin \mathcal{BV}(E)$ voraus und beschaffen uns eine frische Variable z:

$$\begin{array}{lcll}
\exists y \exists x (x = y \wedge E) & \equiv & \exists z \exists x (x = z \wedge E[z/y]) & \alpha\text{-Umbenennung von } y \text{ zu } z \\
& \to & \exists z\, E[z/y][z/x] & \text{Elimination} \\
& \equiv & \exists x\, E[z/y][z/x][x/z] & \alpha\text{-Umbenennung von } z \text{ zu } x \\
& = & \exists x\, E[x/y] & \text{Substitutionsgleichungen}
\end{array}$$

Da Gleichungen ausschließlich durch Elimination beeinflußt werden, verhalten sie sich absolut symmetrisch. Dies erlaubt den Datenfluß in Abhängigkeit von der Berechnungsdynamik zu beschreiben, was wir zur Darstellung der Kontrolle von nicht-strikter funktionaler Berechnung benötigen. Auch Ein- und Ausgabeargumente lassen sich im δ-Kalkül im allgemein nicht statisch determinieren, wie wir an einem Beispiel in Abschnitt 3.2 illustrieren werden.

Das in dieser Arbeit vorgeschlagene Eliminationsaxiom (E) unterscheidet sich technisch wesentlich von demjenigen des γ-Kalküls in der Darstellung von [Smo94c].

Denn dort wird das Axiom

$$\exists x(x = y \wedge E) \;\rightarrow\; E[y/x] \quad \text{falls } x \neq y \text{ und } x \notin \mathcal{BV}(E)$$

verwendet, wobei y nicht als lokal vorausgesetzt ist. Wie wir bereits gesehen haben, ist die Deklaration von y aber notwendig, um die Symmetrie von Gleichungen in Bezug auf Elimination sicherzustellen. In der Tat würde sogar Konfluenz bei Verwendung des alternativen Axioms verloren gehen, wenn man Symmetrie von Gleichungen nicht explizit fordert. Denn dann ließe sich $\exists y(x_1 = y \wedge x_2 = y)$ zu $x_1 = x_2$ und zu $x_2 = x_1$ reduzieren. Aus diesem Grund sind Gleichungen im γ-Kalkül bezüglich Kongruenz als symmetrisch vorausgesetzt. Für die detaillierten Beweise in dieser Arbeit würde Elimination im Stile des γ-Kalküls einen erheblichen Mehraufwand bedeuten.

Wir haben nun den Reduktionsmechanismus des δ-Kalküls vollständig beschrieben, aber seine Definition noch nicht abgeschlossen.

Proposition 3.1.1 *Das Tripel* $(\Delta, \equiv, \rightarrow)$ *ist ein Kalkül.*

Beweis. Die Gültigkeit der Regel (*Congr*) ist äquivalent zur Kalküleigenschaft. □

Wir beschreiben im folgenden die Vermeidung von Inkonsistenzen, was für die Konfluenz des δ-Kalküls notwendig ist. Dazu verwenden wir die Sprechweise, daß x in $x\colon A$ an A gebunden wird gleichbedeutend dazu, daß x in $x\colon A$ eine Referenz auf A ist. Konfluenz kann für solche Ausdrücke verloren gehen, in denen eine Variable an verschiedene Auftreten von Abstraktionen gebunden ist, denn dann hat eine Applikation dieser Variable die freie Auswahl mit welcher Abstraktion sie reduziert.

Wir nennen einen Ausdruck inkonsistent, falls er einen Teilausdruck der Form

$$x\colon A \wedge x\colon A'$$

enthält[2] und konsistent anderenfalls. Ein Ausdruck E heißt zulässig ("admissible"), falls diejenigen F konsistent sind, für die $\overline{x}$ mit $\exists \overline{x} E \rightarrow^* F$ existiert[3]. Zulässigkeit schließt somit alle Ausdrücke aus, die durch wiederholte Anwendung von struktureller Kongruenz, Reduktion oder Abschluß unter Deklaration inkonsistent werden können. Die Menge aller zulässigen Ausdücke bezeichnen wir mit Δ^{ad}.

Es ist plausibel, daß die Zulässigkeit von beliebigen Ausdrücken unentscheidbar ist, denn die Zulässigkeit eines Ausdruckes ist abhängig von der Dynamik eines Turing-vollständigen Systems. Dies ist nicht weiter hinderlich, da wir die Zulässigkeit für alle Ausdrücke unseres Interesses nachweisen können, was wir in Abschnitt 4.3 auch tun werden.

[2] Replikation im Stile des π-Kalküls [Mil91], des κ-Kalküls [Smo94c] oder des frühen Oz-Kalküls [Smo94a] würden bei der Definition von Inkonsistenz Probleme verursachen.

[3] Per Definition in Abschnitt 2.4 ist $\rightarrow^*$ die kleinste transitive Relation, die $\rightarrow$ und $\equiv$ enthält.

Proposition 3.1.2 *Ein Ausdruck E ist zulässig genau dann, wenn $\exists x E$ zulässig ist. Ist $E_1 \wedge E_2$ zulässig, dann sind auch E_1 und E_2 zulässig.*

Beweis. Folgt unmittelbar aus den Definitionen. □

Zulässigkeit ist nicht abgeschlossen unter Komposition und auch nicht unter dem Bilden von Teilausdrücken. So ist zum Beispiel der Ausdruck $x{:}\overline{y}/E$ zulässig, selbst wenn E unzulässig ist.

Proposition 3.1.3 *Die Menge aller zulässigen Ausdrücke Δ^{ad} ist abgeschlossen unter struktureller Kongruenz und Reduktion.*

Beweis. Mit Induktion über Herleitungen von $E \equiv F$ und $E \to F$. □

Definition 3.1.4 *Die Einschränkung von $(\Delta, \equiv, \to)$ auf zulässige Ausdrücke heißt δ-*Kalkül*, in Symbolen $\delta = (\Delta^{ad}, \equiv, \to)$.*

Aus den Propositionen 3.1.3 und 2.5.1 folgt, daß der δ-Kalkül ein Kalkül im Sinne von Definition 2.2.1 ist.

An dieser Stelle ist es instruktiv den δ-Kalkül mit dem γ-Kalkül [Smo94c] und dem ρ-Kalkül [NS94] zu vergleichen. Im Gegensatz zu δ unterscheiden γ und ρ statisch zwischen Variablen und Namen. Jede Abstraktion in γ oder ρ ist mit einem eindeutigen Namen versehen ist. Bei dynamischer Erzeugung einer neuen Abstraktion wird stets auch ein neuer Name erzeugt.

Technisch werden diese Effekte in γ und ρ realisiert, indem syntaktisch ausschließlich Abstraktionen der Form $a{:}\,A$ zulassen werden, wobei a einen Namen denotiert und nicht etwa eine Variable. Ein Ausdruck der Form $x{:}\,A$ ist dann lediglich eine Abkürzungen für $\exists a(x = a \wedge a{:}\,A)$. Eine Inkonsistenz führt zu einer Gleichung der Form $a = b$ mit verschiedenen Namen a und b. Technisch wird dieser Effekt durch (α)-Umbenennung von Namen ermöglicht:

$$\begin{array}{rcl} \exists x(x{:}\,A \wedge x{:}\,A') & = & \exists x(\exists a(x = a \wedge a{:}\,A) \wedge \exists a(x = a \wedge a{:}\,A')) \\ & \equiv_\alpha & \exists x(\exists a(x = a \wedge a{:}\,A) \wedge \exists b(x = b \wedge b{:}\,A')) \\ & \equiv & \exists x \exists a \exists b(x = a \wedge x = b \wedge a{:}\,A \wedge b{:}\,A') \\ & \to_E & \exists a \exists b(b = a \wedge a{:}\,A \wedge b{:}\,A') \end{array}$$

Die Verwendung von α-Umbenennung zur dynamischen Erzeugung neuer Namen wurde auch in [PS93, Ode94] vorgeschlagen.

Wir formulieren nun die Uniformitäts- und Konfluenzeigenschaften des δ-Kalküls. Dank unserer Vorarbeit über Berechnungskalküle in Kapitel 2 können wir den Beweis dieser Eigenschaften auf den Nachweis von drei Vertauschungseigenschaften reduzieren, die in Satz 3.1.7 zusammengefaßt sind.

Satz 3.1.5 *Der δ-Kalkül ist uniform konfluent.*

Daraus folgt insbesondere die Uniformität des Längenmaßes l_δ des δ-Kalküls (Satz 2.3.1). Es gelten allerdings noch weitere Uniformitätseigenschaften, nämlich bezüglich des A-Längenmaßes l_δ^A, das Applikationsschritte in Ableitungen $(E_i)_i$ zählt (vergleiche Kapitel 2.6).

Satz 3.1.6 *Das Längenmaß l_δ und das A-Längenmaß l_δ^A des δ-Kalküls sind uniform, d.h. sie stimmen für alle Berechnungen eines festen Ausdruckes überein.*

Die Sätze 3.1.5 und 3.1.6 sind Konsequenzen des folgenden Satzes 3.1.7, in dem wir feinere Eigenschaften von Applikation und Elimination explizit formulieren, die wir allerdings erst später beweisen werden.

Im folgenden werden wir Axiome frei als binäre Relationen auf Δ auffassen, wobei wir das Relationssymbol des Axioms ignorieren. Zum Beispiel interpretieren wir das Axiom (*Exch*), also $\exists x \exists y\, E \equiv \exists y \exists x\, E$, als Relation aller Paare der Form $(\exists x \exists y\, E,\ \exists y \exists x\, E)$.

Desweiteren definieren wir $\to_R$ für beliebige binäre Relationen R auf Δ als kleinste binäre Relation, die R enthält und (*Congr*), (*Decl*) und (*Comp*) erfüllt. Die Relation $\to_R$ erlaubt also die Anwendung von R unterhalb von Deklaration und Komposition und ist invariant unter struktureller Kongruenz. Insbesondere haben wir somit die Relationen $\to_A$ und $\to_E$ definiert, so daß $\to\ =\ (\to_A \cup \to_E)$ gilt.

Satz 3.1.7 *1. Der Kalkül $(\Delta^{ad}, \equiv, \to_A)$ ist uniform konfluent.*

2. Der Kalkül $(\Delta, \equiv, \to_E)$ ist uniform konfluent und terminierend.

3. Die Relationen $\to_E$ und $\to_A$ vertauschen.

Alle behaupteten Vertauschungsaussagen in diesem Satz werden wir in Abschnitt 3.5 beweisen, wozu wir die Basen aus Abschnitt 3.4 verwenden werden. Die Terminierung von $\to_E$ ist offensichtlich; denn jeder Eliminationsschritt verkleinert die Masse eines Ausdrucks echt um eine Gleichung und eine Deklaration. Dabei ist zu beachten, daß die Masse von Ausdrücken invariant unter struktureller Kongruenz ist.

Beweis der Sätze 3.1.5 und 3.1.6. Der δ-Kalkül ist die Vereinigung von $(\Delta^{ad}, \equiv, \to_E)$ und $(\Delta^{ad}, \equiv, \to_A)$. Nach Satz 3.1.7 sind beide Kalküle uniform konfluent und ihre Reduktionen vertauschen. Wir können also Proposition 2.5.3 anwenden, woraus die uniforme Konfluenz des δ-Kalküls, d.h. Satz 3.1.5, folgt. Die Uniformität des Längenmaßes folgt aus Satz 2.3.1 und uniformer Konfluenz. Desweiteren ist $(\to_A, \to_E)$ eine Zerlegung auf Δ^{ad}. Somit folgt die Uniformität des A-Längenmaßes aus Satz 2.6.2 und wir haben Satz 3.1.6 gezeigt. □

3.2 Beispiele

Wir illustrieren den δ-Kalkül anhand einiger Beispiele. Diese weisen auf syntaktische Feinheiten hin, zeigen das Abarbeiten von Referenzketten durch Elimination, illustrieren ungerichteten Datenfluß, stellen Bäume dar und diskutieren Inkonsistenzen.

Im folgenden sei Id stets die δ-Identität $y\,z/z = y$. In unserem ersten Beispiel applizieren wir Id auf sich selbst[4].

$$\begin{array}{lll} \exists u \exists v\,(u\colon Id \wedge v\colon Id \wedge u\,v\,out) & \rightarrow_A & \exists u \exists v\,(u\colon Id \wedge v\colon Id \wedge out = v) \\ & \equiv & \exists v\,(out = v \wedge v\colon Id) \wedge \exists u\,u\colon Id \end{array}$$

Dies zeigt zwei syntaktische Feinheiten. Erstens können wir nicht alle Gleichungen zwischen Variablen entfernen, solange globale Variablen im Spiel sind. Dies ist im Beispiel nicht wesentlich, da eine Elimination von v keine weiteren Applikationsschritte ermöglichen würde. Gleiches gilt für alle anderen Beispiele in dieser Arbeit.

Der zweite Effekt ist, daß lokale Abstraktionen zurückbleiben, die den Fortgang der Berechnung nicht mehr beeinflussen können. Dies mag für konkrete Beispiele lästig sein, ist allerdings nicht wesentlich für Terminierung und Komplexität, wie wir in Abschnitt 3.3 zeigen werden.

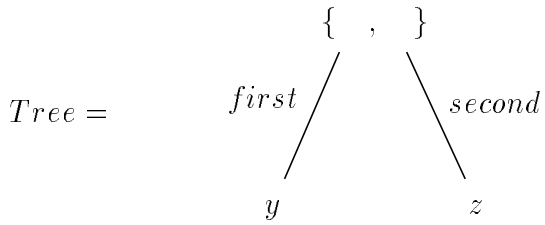
Das nächste Beispiel zeigt, wie Elimination Ketten von lokalen Referenzen verkürzt und weist im Vergleich zu allen vorangehenden Beispielen einen anderen Datenfluß auf. Wir wenden die δ-Identität auf out an und legen das Resultat dieser Applikation durch y fest, welches über eine Referenzkette an die δ-Identität gebunden ist.

$$\begin{array}{lll} \exists x \exists y \exists z\,(x\colon Id \wedge x = y \wedge y = z \wedge z\,out\,y) & \rightarrow_E & \exists x \exists z\,(x\colon Id \wedge x = z \wedge z\,out\,z) \\ & \rightarrow_E & \exists z\,(z\colon Id \wedge z\,out\,z) \\ & \rightarrow_A & \exists z\,(z\colon Id \wedge z = out) \end{array}$$

In diesem Fall fungiert das erste Argument als Ausgabe und das zweite als Eingabe.

Nun wollen wir Bäume im δ-Kalkül darstellen, was wir in Analogie zu bekannten Kodierungen von Datenstrukturen im λ-Kalkül [Bar90] tun. Als Beispiel beschreiben wir das folgende Bild im δ-Kalkül:

[4] Bezüglich der Einbettung des strikten λ-Kalküls entspricht dies der Berechnung von $out = II$.



Dazu stellen wir uns den Baum als funktionale Prozedur vor, die einen Selektor einliest und den selektierten Unterbaum ausgibt.

Als Selektoren verwenden wir die folgenden beiden Abstraktionen $First$ und $Second$[5]:

$$x\,y\,z\,/\,z = x \qquad \text{und} \qquad x\,y\,z\,/\,z = y\,.$$

Den Baum $Tree$ im obigen Bild beschreiben wir nun durch folgende Abstraktion:

$$sel\,val\,/\,sel\,y\,z\,val\,.$$

In einem passenden Kontext können wir nun sowohl "lesend" als auch "schreibend" auf die Unterbäume von $Tree$ zugreifen. Um diese Aussage formal zu erfassen führen wir neue Metasyntax für Kontexte ein, die wir auch in den folgenden Kapiteln benötigen werden. Unter einem Kontext T verstehen wir einen δ-Ausdruck mit einem Loch $\bullet$; Mit $T[\![E]\!]$ bezeichnen wir denjenigen Ausdruck, der durch textuelle Ersetzung des Loches in T durch E entsteht. Wir sagen, daß $E \to F$ im Kontext T gilt, falls $T[\![E]\!] \to T[\![F]\!]$. Für eine ausführliche Definition verweisen wir auf Abschnitt 5.2.2.

Im Zusammenhang mit der Darstellung unseres Baumes $Tree$ interessieren wir uns für den folgenden Kontext, den wir T nennen:

$$\exists tree \exists y \exists first \exists second\,(tree{:}\,Tree \wedge first{:}\,First \wedge second{:}\,Second \wedge \bullet)\,.$$

Nun können wir zeigen, wie wir mit Unterbäume von $Tree$ rechnen können. Dazu betrachten wir die folgende Reduktion im Kontext T:

$$\begin{array}{lll}
\exists id(id{:}\,Id \wedge tree\,first\,id \wedge tree\,first\,out) & & \\
\quad \to_A \quad \exists id(id{:}\,Id \wedge first\,y\,z\,id \wedge tree\,first\,out) & & \\
\quad \to_A \quad \exists id(id{:}\,Id \wedge id = y \wedge tree\,first\,out) & & \text{"schreibend"} \\
\quad \to_E \quad y{:}\,Id \wedge tree\,first\,out & & \\
\quad \to_A \quad y{:}\,Id \wedge first\,y\,z\,out & & \\
\quad \to_A \quad y{:}\,Id \wedge out = y & & \text{"lesend"}
\end{array}$$

Die ersten beiden Applikationen binden y an Id, die nachfolgenden binden out an die von y referierte Abstraktion, also ebenfalls an Id. Wir hätten die Reihenfolge

[5]Im λ-Kalkül würden wir analog $First = \lambda x.\lambda y.\ x$ und $Second = \lambda x.\lambda y.\ y$ verwenden.

der Zugriffe aber auch ebensogut vertauschen können. Konsistenz bleibt deshalb erhalten, weil wir nur einmal "schreibend" auf y zugreifen. Ein weiterer schreibender Zugriff könnte zur Inkonsistenz führen und zwar dann, wenn dadurch y an eine andere Abstraktion oder ein anderes Auftreten von Id gebunden wird.

Im γ-Kalkül [Smo94c] lassen sich sogar "Records" mit "Adjunktion" darstellen, wobei Namen als Schlüssel dienen. Dazu ist ein Konditional notwendig, das nicht nur auf die Gleichheit, sondern auch auf die Verschiedenheit von Namen testen kann. Erstaunlicherweise können wir im δ-Kalkül auch ohne Namen einen sinnvollen Test auf Verschiedenheit formulieren. Dazu ist es hinreichend ein Konditional mit folgenden Reduktionsregeln zu erlauben:

$$\begin{array}{rcl} \mathsf{if}\ x = x\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi} & \to & E \\ x\colon A \wedge y\colon A' \wedge \mathsf{if}\ x = y\ \mathsf{then}\ E\ \mathsf{else}\ F\ \mathsf{fi} & \to & F \wedge x\colon A \wedge y\colon A' \end{array}$$

Die Regel für den *else*-Fall macht Sinn, da Zulässigkeit im Kontext von $x\colon A \wedge y\colon A'$ sicherstellt, daß x und y niemals gleichgesetzt werden können. Auch die uniforme Konfluenz des δ-Kalküls würde durch Konditionale nicht beeinträchtigt.

3.3 Annullierung

Bisher haben wir uns bei der Darstellung des δ-Kalküls ausschließlich um Komplexität und Terminierung gekümmert. Nun fragen wir nach dem benötigten Speicherplatz. Eine geeignete Abschätzung liefert die maximale Größe eines Ausdruckes der betrachteten Berechnung. Dies ist jedoch nicht realistisch, da wir im δ-Kalkül lokale Abstraktionen *nicht* entfernen können, aber der entsprechende Speicherplatz in konkreten Systemen durch Speicherbereinigung ("Garbage Collection") wiederverwendet werden kann.

Aus diesem Grund schlagen wir eine Erweiterung der Reduktion des δ-Kalküls um Annullierung[6] (G) vor, wobei wir $\overline{y}\colon \overline{A}$ als Abkürzung für eine endliche Komposition von Abstraktionen verwenden und $\top$ für die leere Komposition:

$$(G) \qquad \exists \overline{x}\,\overline{y}\colon \overline{A} \ \to\ \top \qquad \text{falls } \mathcal{V}(\overline{y}) \subseteq \mathcal{V}(\overline{x}) \text{ und } \mathcal{V}(\overline{x}) \neq \emptyset\ .$$

Insbesondere erlaubt diese Regel, überflüssige Deklarationen wegzuwerfen. Denn unter der Bedingung $\mathcal{V}(\overline{x}) \cap \mathcal{FV}(E) = \emptyset$ können wir folgendermaßen annullieren:

$$\exists \overline{x}\, E \ \equiv\ \exists \overline{x}(\top \wedge E) \ \equiv\ (\exists \overline{x}\, \top) \wedge E \ \to_G\ \top \wedge E \ \equiv\ E\ .$$

Annullierung wird auch im Oz-Kalkül [Smo94a, Smo94b] verwendet, nämlich zur Realisierung von tiefen Wächtern ("deep Guards"). Tiefe Wächter beschreiben

[6] (G) steht für "Garbage Collection".

die Reduktionbedingung eines relationalen Konditionals durch eine beliebige, von der Außenwelt enkapsulierte Berechnung. Auch in [AFM+95] wird Annullierung vorgeschlagen. Dort dient sie dazu, Ausdrücke wegzuwerfen, die bei nicht-strikter funktionaler Berechnung nicht benötigt werden.

Wir beginnen nun mit dem Nachweis, daß Annullierung (G) im Falle des δ-Kalküls Terminierung nicht beeinflußt und auch Komplexität und Uniformität in einem geeigneten Sinne erhalten bleiben. Dazu müssen wir natürlich zuerst sicherstellen, daß Annullierung keine neuen Inkonsistenzen erzeugt.

Proposition 3.3.1 *Die Relation $\rightarrow_G$ erhält Zulässigkeit.*

Zum Beweis wollen wir die leider relativ schwachen Abschlußeigenschaften von Zulässigkeit verwenden, die wir in den Propositionen 3.1.2 und 3.1.3 formuliert haben. Der Leser kann sich davon überzeugen, daß eine naive Induktion über Herleitungen von $E \rightarrow_G F$ an der Regel (*Comp*) scheitert. Das folgende Lemma löst dieses Problem.

Lemma 3.3.2 *Sei E zulässig und gelte $E \rightarrow_G F$. Dann existieren $\overline{z}$, E_1 und F_1 mit $E \equiv \exists\overline{z}(E_1 \wedge F_1)$, $E_1 \rightarrow_G \top$ und $\exists\overline{z}F_1 \equiv F$.*

Beweis. Mit Induktion über Herleitungen von $E \rightarrow_G F$. □

Beweis von Proposition 3.3.1. Sei E zulässig, gelte $E \rightarrow_G F$ und seien $\overline{z}$, E_1 und F_1 wie in Lemma 3.3.2. Da E zulässig ist, ist auch $\exists\overline{z}(E_1 \wedge F_1)$ zulässig (Proposition 3.1.3). Wegen Proposition 3.1.2 sind $E_1 \wedge F_1$, F_1, und $\exists\overline{z}F_1$ zulässig. Wiederum nach Proposition 3.1.3 folgt die Zulässigkeit von F. □

Satz 3.3.3 *1. Der Kalkül $(\Delta, \equiv, \rightarrow_G)$ ist stark konfluent und terminierend.*

2. Die Relation $\rightarrow_G$ vertauscht mit $\rightarrow_A$ und $\rightarrow_E$.

3. Schritte in $\rightarrow_G$ erhalten Irreduzibilität bezüglich $\rightarrow_A$ und $\rightarrow_E$.

Die Vertauschungseigenschaften der beiden ersten Teile beweisen wir in Abschnitt 3.5. Einen Beweis des dritten Teils skizzieren wir am Ende von Abschnitt 3.4. Die Terminierung von $\rightarrow_G$ ist offensichtlich, da jede Anwendung von $\rightarrow_G$ die Masse von Ausdrücken zumindest um eine Deklaration verkleinert.

Als δ-Kalkül mit Annullierung bezeichnen wir $\delta' = (\Delta^{ad}, \equiv, (\rightarrow_A \cup \rightarrow_E \cup \rightarrow_G))$. Das A-Längenmaß $l^A_{\delta'}$ zählt Applikationsschritte in Ableitungen von δ'; das AE-Längenmaß $l^{AE}_{\delta'}$ zählt Applikations- und Eliminationsschritte.

Satz 3.3.4 *Der δ-Kalkül mit Annullierung δ' ist konfluent und die Komplexitätsmaße $l^A_{\delta'}$ und $l^{AE}_{\delta'}$ sind uniform. Eingeschränkt auf Berechnungen des δ-Kalküls gilt $l^A_\delta = l^A_{\delta'}$ und $l_\delta = l^{AE}_{\delta'}$.*

Beweis. Der δ-Kalkül mit Annullierung ist die Vereinigung von drei konfluenten Kalkülen, deren Reduktionen paarweise vertauschen (Satz 3.3.3 und Satz 3.1.7). Somit folgt die Konfluenz von δ' aus dem Lemma von Hindley-Rosen, d.h. Proposition 2.5.4.

Aus Proposition 3.3.1 und den Sätzen 3.3.3 und 3.1.7 folgt, daß $(\to_A, (\to_E \cup \to_G))$ und $((\to_A \cup \to_E), \to_G)$ Zerlegungen auf Δ^{ad} sind. Somit können wir die Uniformität von $l^A_{\delta'}$ und von $l^{AE}_{\delta'}$ aus Satz 2.6.2 schließen.

Die Gleichheit $l_\delta = l^{AE}_{\delta'}$ erhalten wir aus Satz 2.7.5, wobei wir dessen Parameter wie folgt festsetzen, also folgende Zerlegungen in Reduktion und Kontrolle betrachten:

$$\begin{array}{rclcrcl} (K, \equiv) & = & (\Delta^{ad}, \equiv)\,, & & \to_r & = & (\to_A \cup \to_E)\,, \\ \to_c & = & \equiv\,, & & \to_d & = & \to_G\ . \end{array}$$

Die erste Voraussetzung von Satz 2.7.5 folgt unmittelbar aus Satz 3.3.3 Teil 3 und die zweite gilt trivialerweise.

Zum Nachweis von $l^A_\delta = l^A_{\delta'}$ verwenden wir wiederum Satz 2.7.5. Diesmal setzen wir die Parameter folgendermaßen:

$$\begin{array}{rclcrcl} (K, \equiv) & = & (\Delta^{ad}, \equiv)\,, & & \to_r & = & \to_A\,, \\ \to_c & = & \to_E\,, & & \to_d & = & (\to_E \cup \to_G)\,. \end{array}$$

Die zweite Voraussetzung von Satz 2.7.5 ist wiederum trivial. Die erste fordert, daß $(\to_E \cup \to_G)$ die Irreduzibilität bezüglich $(\to_A \cup \to_E)$ erhält. Dies ist äquivalent dazu, daß $\to_G$ die Irreduzibilität von $(\to_A \cup \to_E)$ erhält, was wiederum von Satz 3.3.3 Teil 3 sichergestellt wird. □

3.4 Basen des δ-Kalküls

In diesem Abschnitt definieren wir Definitions- und Reduktionsbasen des δ-Kalküls, die wir zum Nachweis der behaupteten Vertauschungseigenschaften benötigen. Die Beweise der in diesem Abschnitt behaupteten Aussagen werden wir in Kapitel 5 führen. Dadurch faktorisieren wir unsere Konfluenzbeweise, so daß wir langwierige und technisch unangenehme Betrachtungen im Zusammenhang mit α-standardisierten Pränexnormalformen auslagern können. Der Leser kann sich aber bei Bedarf von deren Vollständigkeit überzeugen.

Wir vereinbaren zuerst eine Sammlung an Notationen für spezielle binäre Relationen auf Ausdrücken. Sei R eine binäre Relation auf Δ. Dann bezeichnen wir mit

$\equiv_R$ die kleinste Kongruenz auf Δ, die R enthält. So ist zum Beispiel $\equiv_\alpha$ die kleinste Kongruenz, die das Axiom (α) enthält. Desweiteren gilt $\equiv \; = \; \equiv_{\alpha \cup ACI \cup Exch \cup Scope}$. Die Relation $\bar{\to}_R$ ist die kleinste binäre Relation, die R enthält und die Regeln $(Decl)$, $(Comp)$ und $(Comp')$ erfüllt:

$$(Comp') \quad \frac{E \;\to\; F}{G \wedge E \;\to\; G \wedge F} \quad .$$

Somit erlaubt $\bar{\to}_R$ die Anwendung von R unterhalb von Deklaration und Komposition. Die zu $(Comp)$ symmetrische Regel $(Comp')$ ist erforderlich, da Symmetrie der Komposition hier nicht durch Abschluß unter struktureller Kongruenz sichergestellt wird. Dies ist gerade der Unterschied zwischen $\to_R$ und $\bar{\to}_R$.

Ein Ausdruck E heißt α-standardisiert, falls zu keiner Variable in E mehrere Variablenbinder in E existieren, und für alle Teilausdrücke F von E die Mengen der gebundenen und freien Variablen von F disjunkt sind. Die Menge der α-standardisierten Ausdrücke bezeichnen wir mit Δ^α. Die Relation $\overset{s}{\Rightarrow}_\alpha$ ist die kleinste binäre Relation auf Δ, die gebundene Variablen unter Einführung frischer Variablen umbenennt; jede solche Umbenennung ist natürlich konsistent. So gilt zum Beispiel $x{:}y/\exists z\, z = x \;\overset{s}{\Rightarrow}_\alpha\; x{:}y/\exists z'\, z' = x$ genau dann, wenn z' verschieden von x, y und z ist. Desweiteren definieren wir die Kongruenz $\equiv_1$ durch:

$$\equiv_1 \;\; = \;\; \equiv_{ACI \cup Exch \cup Scope} \quad .$$

Die Beweise für alle folgenden Sätze dieses Abschnittes werden wir in Kapitel 5 erbringen.

Satz 3.4.1 *Das Tripel* $(\Delta^\alpha, (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ *ist eine Definitionsbasis von* $(\Delta, \equiv)$. *Eingeschränkt auf* $\Delta^\alpha \times \Delta^\alpha$ *vertauscht* $\overset{s}{\Rightarrow}^*_\alpha$ *mit* $\equiv_1$ *und stimmt mit* ${}^*_\alpha\overset{s}{\Leftarrow}$ *überein.*

Aus diesem Satz lassen sich einige Definitionsbasen des δ-Kalkül durch Vereinigung und Einschränkung ableiten, da Δ^{ad} abgeschlossen unter $\equiv_1$ und $\overset{s}{\Rightarrow}_\alpha$ ist. So ist $(\Delta^\alpha \cap \Delta^{ad}, (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ eine Definitionsbasis des δ-Kalküls. Wegen der Vertauschungseigenschaften gilt außerdem:

$$(\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^* \;\; = \;\; (\equiv_1 \circ \overset{s}{\Rightarrow}^*_\alpha) \;\; = \;\; (\equiv_1 \circ {}^*_\alpha\overset{s}{\Leftarrow}) \;\; = \;\; ({}^*_\alpha\overset{s}{\Leftarrow} \circ \equiv_1) \;\; = \;\; (\overset{s}{\Rightarrow}^*_\alpha \circ \equiv_1) \, .$$

Satz 3.4.2 *Das Tripel* $(\Delta^\alpha, \equiv_1, \bar{\to}_E)$ *ist eine Reduktionsbasis von* $(\Delta, \equiv, \to_E)$, *das Tripel* $(\Delta^\alpha, \equiv_1, \bar{\to}_A)$ *ist eine Reduktionsbasis von* $(\Delta, \equiv, \to_A)$ *und das Tripel* $(\Delta^\alpha, \equiv_1, \bar{\to}_G)$ *ist eine Reduktionsbasis von* $(\Delta, \equiv, \to_G)$.

Aus diesem Satz lassen sich Reduktionsbasen für den δ-Kalkül mit und ohne Annullierung ableiten: Das Tripel $(\Delta^{\alpha} \cap \Delta^{ad}, \equiv_1, (\bar{\rightarrow}_E \cup \bar{\rightarrow}_A))$ ist eine Reduktionsbasis von δ, und $(\Delta^{\alpha} \cap \Delta^{ad}, \equiv_1, (\bar{\rightarrow}_E \cup \bar{\rightarrow}_A \cup \bar{\rightarrow}_G))$ ist eine Reduktionsbasis von δ'.

Im nächsten Schritt definieren wir Pränexnormalformen (PNFs) von Ausdrücken in Δ, und stellen darauf basierende Definitions- und Reduktionsbasen vor. Eine PNF ist ein Ausdruck in Δ, in dem alle Deklarationen so weit wie möglich nach außen bewegt sind (*Scope*). Die formale Definition ist in Abbildung 3.4 gegeben, in Rekursion mit der Definition von zwei weiteren Klassen von Ausdrücken, Molekül und chemische Lösungen (diese Bezeichnungen verwenden wir motiviert durch [BB90]).

B	$::=$	$x{:}\overline{y}/D \mid x\,\overline{y} \mid x = y$	Moleküle
C	$::=$	$\top \mid B \mid C \wedge C$	Chemische Lösungen
D	$::=$	$C \mid \exists x\, D$	PNFs

Abbildung 3.4: PNFs

Ein Molekül ist eine Abstraktion über einer PNF, eine Applikation oder eine Gleichung. Eine chemische Lösung ist eine möglicherweise leere Komposition von Molekülen oder Nullen und eine PNF besteht aus einem Präfix von Deklarationen und einer chemischen Lösung.

Eine Normalform ist eine α-standardisierte PNF. Die Menge aller Normalformen bezeichnen wir mit Δ^{nf}. In Abbildung 3.5 definieren wir drei Relationen $\bar{\rightharpoonup}_{A_t}$, $\bar{\rightharpoonup}_{E_t}$ und $\bar{\rightharpoonup}_{G_t}$ auf $\Delta^{nf} \times \Delta$, für Applikation, Elimination und Annullierung auf Normalformen. Wir verwenden andere Pfeile für diese Relationen, da sie nicht invariant unter Komposition sind und wir ihre Invarianz unter Deklaration nicht explizit fordern.

$$\exists \overline{u}(x{:}\overline{y}/F \wedge x\,\overline{z} \wedge E) \;\; \bar{\rightharpoonup}_{A_t} \;\; \exists \overline{u}(x{:}\overline{y}/F \wedge F[\overline{z}/\overline{y}] \wedge E)$$

$$\exists \overline{u} \exists x(x = y \wedge E) \;\; \bar{\rightharpoonup}_{E_t} \;\; \exists \overline{u}\, E[y/x] \qquad \text{falls } y \in \mathcal{V}(\overline{u})$$

$$\exists \overline{u} \exists \overline{x}(\overline{y}{:}\,\overline{A} \wedge E) \;\; \bar{\rightharpoonup}_{G_t} \;\; \exists \overline{u}\, E \qquad \text{falls } \mathcal{V}(\overline{x}) \cap \mathcal{FV}(E) = \emptyset,\ \mathcal{V}(\overline{y}) \subseteq \mathcal{V}(\overline{x}) \text{ und } \mathcal{V}(\overline{x}) \neq \emptyset$$

Abbildung 3.5: Reduktion auf Normalformen

Desweiteren definieren wir die Kongruenz $\equiv_2$ auf Δ durch $\equiv_2 \;=\; \equiv_{ACI \cup Exch}$.

Satz 3.4.3 *Das Tripel* $(\Delta^{nf}, (\equiv_2 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ *ist eine Definitionsbasis von* $(\Delta, \equiv)$. *Eingeschränkt auf* $\Delta^{nf} \times \Delta^{nf}$ *vertauscht* $\overset{s}{\Rightarrow}^*_\alpha$ *mit* $\equiv_2$ *und stimmt mit* ${}^*_\alpha\overset{s}{\Leftarrow}$ *überein.*

Da Δ^{nf} abgeschlossen unter $\equiv_2$ und $\overset{s}{\Rightarrow}_\alpha$ ist, ist $(\Delta^{nf} \cap \Delta^{ad}, (\equiv_2 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ eine Definitionsbasis des δ-Kalküls. Desweiteren gilt eingeschränkt auf zulässige Normalformen:

$$(\equiv_2 \cup \overset{s}{\Rightarrow}_\alpha)^* \quad = \quad (\equiv_2 \circ \overset{s}{\Rightarrow}^*_\alpha) \quad = \quad (\equiv_2 \circ {}^*_\alpha\overset{s}{\Leftarrow}) \quad = \quad ({}^*_\alpha\overset{s}{\Leftarrow} \circ \equiv_2) \quad = \quad (\overset{s}{\Rightarrow}^*_\alpha \circ \equiv_2)\,.$$

Satz 3.4.4 *Das Tripel* $(\Delta^{nf}, \equiv_2, \bar{\rightarrow}_{A_t})$ *ist eine Reduktionsbasis von* $(\Delta, \equiv, \rightarrow_A)$, *das Tripel* $(\Delta^{nf}, \equiv_2, \bar{\rightarrow}_{E_t})$ *ist eine Reduktionsbasis von* $(\Delta, \equiv, \rightarrow_E)$, *und das Tripel* $(\Delta^{nf}, \equiv_2, \bar{\rightarrow}_{G_t})$ *ist eine Reduktionsbasis von* $(\Delta, \equiv, \rightarrow_G)$.

Daraus können wir Reduktionsbasen für δ und δ' ableiten: Das Tripel $(\Delta^{nf} \cap \Delta^{ad}, \equiv_2, (\bar{\rightarrow}_{E_t} \cup \bar{\rightarrow}_{A_t}))$ ist eine Reduktionsbasis des δ-Kalküls und das Tripel $(\Delta^{nf} \cap \Delta^{ad}, \equiv_2, (\bar{\rightarrow}_{E_t} \cup \bar{\rightarrow}_{A_t} \cup \bar{\rightarrow}_{G_t}))$ eine Reduktionsbasis des δ-Kalküls mit Annullierung.

Als Anwendung der Reduktionsbasen aus Satz 3.4.4 und von Proposition 2.8.9 können wir nun Reduzierbarkeit im δ-Kalkül entscheiden: Sei E ein δ-Ausdruck und F eine Normalform mit $E \equiv F$. Dann E ist genau dann reduzierbar bezüglich $\rightarrow_A$, wenn F bezüglich $(\equiv_2 \circ \bar{\rightarrow}_{A_t})$ reduzierbar ist. Die analogen Aussagen für Elimination und Annullierung gelten ebenfalls.

Betrachten wir zum Beispiel den Ausdruck $E \equiv x{:}y/\top \wedge \exists x\, x\, y$, wobei xyz linear ist, und F mit:

$$F \;\equiv\; \exists z\,(x{:}y/\top \wedge z\, y)\,.$$

Dann ist F eine Normalform mit $E \equiv F$. Irreduzibilität von F bezüglich $(\equiv_2 \circ \bar{\rightarrow}_{E_t})$ und $(\equiv_2 \circ \bar{\rightarrow}_{A_t})$ ist leicht zu zeigen. Dazu nützen wir aus, daß sich die Kongruenzklasse von F in $\equiv_2$ mit einem Paar von Multimengen identifizieren läßt[7], die aus Variablendeklarationen bzw. Molekülen bestehen. Der Ausdruck F ist irreduzibel bezüglich $(\equiv_2 \circ \bar{\rightarrow}_{E_t})$, da die Moleküle von F keine Gleichung enthalten und irreduzibel bezüglich $(\equiv_2 \circ \bar{\rightarrow}_{A_t})$, da unter den Molekülen von F keine Abstraktion und Applikation mit der gleichen Referenz existieren.

Mit einem ähnlichen Argument können wir auch Satz 3.3.3 Teil 3 beweisen, nämlich, daß die Relation $\rightarrow_G$ Irreduzibilität bezüglich $\rightarrow_A$ und $\rightarrow_E$ erhält.

[7]Diese Tatsache werden wir in Abschnitt 3.5.1 formalisieren.

3.5 Beweis der uniformen Konfluenz

Wir wollen nun die uniforme Konfluenz des δ-Kalküls mit und ohne Annullierung nachweisen. Dazu müssen wir Satz 3.1.7 und Satz 3.3.3 zeigen. Als wesentliches Hilfsmittel verwenden wie die auf Normalformen basierenden Reduktionsbasen von Satz 3.4.4.

3.5.1 Vorbereitungen

Um die Details des eigentlichen Beweises zu rechtfertigen, benötigen wir einfache Eigenschaften von Substitutionen und eine formale Analyse von $\equiv_2$ als Multimengenrelation. Die folgenden Eigenschaften sind offensichtlich und können vorerst übersprungen und bei Bedarf gelesen werden.

Analyse von $\equiv_2$ auf Normalformen.

Intuitiv können wir die Kongruenzklassen einer Normalform modulo $\equiv_2$ als Paar, bestehend aus Präfix von Deklarationen und einer chemischen Lösung auffassen. Die Äquivalenzklasse einer chemischen Lösung modulo $\equiv_2$ ist eine Multimenge von Molekülen und die Äquivalenzklasse eines Präfixes in $\equiv_2$ ist eine Menge von Variablen. Denn wegen α-Standardisierung kommen in Präfixen von Normalformen Variablen nicht mehrfach vor.

Um diese Intuitionen zu formalisieren, definieren wir eine Äquivalenzrelation $\approx$ auf chemischen Lösungen, sowie eine analoge Äquivalenzrelation $\approx$ auf Präfixen, deren Klassen sich auch formal wie Multimengen behandeln lassen. Wir zeigen, wie sich das Rechnen mit $\equiv_2$ auf das Rechnen mit diesen beiden Äquivalenzrelationen zurückführen läßt, wodurch wir im folgenden mit $\equiv_2$ exakt im Sinne unserer Intuition umgehen können. Der Konfluenzbeweis sollte für einen versierten Leser auch dann verstehbar sein, wenn er die formalen Details dieses vorbereitenden Abschnittes überspringt.

Wir definieren $\approx$ auf chemischen Lösungen als kleinste Äquivalenzrelation, die (ACI) enthält und die folgenden Regeln erfüllt:

$$\frac{D \equiv_2 D'}{x{:}\overline{y}/D \approx x{:}\overline{y}/D'} \qquad \frac{B_1 \approx B_2}{C \wedge B_1 \approx C \wedge B_2}$$

Somit läßt $\approx$ Anwendungen von (ACI) in beliebigen Positionen zu, aber Anwendungen von $\equiv_2$ nur in Rümpfen von Abstraktionen. Auf Präfixen definieren wir $\exists\overline{x} \approx \exists\overline{y}$ durch $\exists\overline{x}\top \equiv_{Exch} \exists\overline{y}\top$.

Proposition 3.5.1 *Für alle chemischen Lösungen C_1 und C_2 gilt:*

$$\exists\overline{x_1}C_1 \equiv_2 \exists\overline{x_2}C_2 \quad gdw \quad \exists\overline{x_1} \approx \exists\overline{x_2} \text{ und } C_1 \approx C_2\,.$$

Beweis. Die Implikation von rechts nach links folgt mit Induktion über Herleitungen von $\exists\overline{x_1}\top \equiv_{Exch} \exists\overline{x_2}\top$. Die Implikation von links nach rechts können wir mit Induktion über Herleitungen von $\exists\overline{x_1}C_1 \equiv_2 \exists\overline{x_2}C_2$ nachweisen. Wir spielen zuerst den Fall für das Axiom der Reflexivität, $E \equiv_2 E$, durch. Falls $E = \exists\overline{x_1}C_1$ und $E = \exists\overline{x_2}C_2$, dann folgt aus der Definition einer chemischen Lösung $\overline{x_1} = \overline{x_2}$ und $C_1 = C_2$. Da beide Varianten von $\approx$ Äquivalenzrelationen sind, folgt $\exists\overline{x_1} \approx \exists\overline{x_2}$, sowie $C_1 \approx C_2$.

Wir betrachten nun das Axiom $E \wedge \top \equiv_2 E$. Dann gilt $E \wedge \top = \exists\overline{x_1}C_1$ und $E = \exists\overline{x_2}C_2$. Aus der ersten Gleichung folgt, daß $\overline{x_1}$ die leere Folge sein muß und $E \wedge \top = C_1$ gilt. Durch Einsetzen der zweiten Gleichung erhalten wir $\exists\overline{x_2}C_2 \wedge \top = C_1$. Da C_1 eine chemische Lösung ist, muß auch die Folge $\overline{x_2}$ leer sein und $C_2 \wedge \top = C_1$ gelten. Daraus folgt $\exists\overline{x_1} \approx \exists\overline{x_2}$ und $C_2 \approx C_1$.

Die Betrachtungen für die Assoziativität und Kommutativität von $\wedge$, sowie für die Symmetrie von Äquivalenzrelationen sind einfach. Im Fall der der Transitivität benötigen wir, daß die Menge aller PNFs unter $\equiv_2$ abgeschlossen ist. Die Argumentationen für die Regeln einer Kongruenz sind einfach. □

Nun wollen wir zeigen, daß wir mit Kongruenzklassen von chemischen Lösungen modulo $\approx$ wie mit einer Multimenge von Molekülen rechnen können. Um technische Schwierigkeiten mit der Null $\top$ zu lösen, definieren wir $\approx_1$ als kleinste Äquivalenzrelation auf chemischen Lösungen, die die beiden Regeln von $\approx$ erfüllt und das Axiom (AC) für Assoziativität und Kommutativität enthält.

$$(AC) \qquad (C_1 \wedge C_2) \wedge C_3 \approx C_1 \wedge (C_2 \wedge C_3) \qquad C_1 \wedge C_2 \approx C_2 \wedge C_1$$

Die Relationen $\approx$ und $\approx_1$ unterscheiden sich also durch die Gültigkeit des Axioms $E \wedge \top \equiv \top$.

Unter einer Komposition von Molekülen verstehen wir eine chemische Lösung, die durch

$$C ::= B \mid C \wedge C$$

erzeugt wird, also eine chemische Lösung, in der $\top$ nicht vorkommt.

Lemma 3.5.2 *Eingeschränkt auf Kompositionen von Molekülen stimmen $\approx$ und $\approx_1$ überein.*

Beweis. Die Inklusion $\approx_1 \subseteq \approx$ gilt offensichtlich für alle chemischen Lösungen. Wir skizzieren nun den Beweis der umgekehrten Inklusion, eingeschränkt auf Kompositionen von Molekülen. Es ist einfach eine Funktion *Filter* zu definieren, die

chemische Lösungen auf Kompositionen von Molekülen oder $\top$ abbildet, indem sie alle überflüssigen Nullen streicht, und $Filter(\top) = \top$ setzt. Dann läßt sich mit Induktion über Herleitungen von $C_1 \approx C_2$ für alle chemische Lösungen C_1, C_2 zeigen:

1. Aus $C_1 \approx C_2$ folgt $Filter(C_1) \approx_1 Filter(C_2)$.

Mit struktureller Induktion über Kompositionen von Molekülen erhalten wir:

2. Ist C eine Komposition von Molekülen, dann gilt $C = Filter(C)$.

Sind nun C_1 und C_2 Kompositionen von Molekülen mit $C_1 \approx C_2$, dann folgt:

$$C_1 \;=\; Filter(C_1) \;\approx_1\; Filter(C_2) \;=\; C_2\,. \qquad \square$$

Lemma 3.5.3 *Es gilt $B \approx x{:}\overline{y}/D$ genau dann, wenn D' existiert mit $B = x{:}\overline{y}/D'$ und $D \equiv_2 D'$.*

Beweis. Mit Induktion über Herleitungen von $B \approx_1 x{:}\overline{y}/D$ läßt sich das Lemma für $\approx_1$ anstelle von $\approx$ zeigen, was nach Lemma 3.5.2 hinreichend ist. $\square$

Lemma 3.5.4 *Ist B_1 eine Gleichung oder eine Applikation, dann gilt $B_1 \approx B_2$ genau dann, wenn $B_1 = B_2$.*

Beweis. Die analoge Aussage für $\approx_1$ anstelle von $\approx$ folgt mit Induktion über Herleitungen von $B_1 \approx_1 B_2$ und dies ist wegen Lemma 3.5.2 hinreichend. $\square$

Lemma 3.5.5 *Für jedes C gilt $C \approx \top$ oder es existieren $B_1, \ldots, B_n$ mit $C \approx B_1 \wedge \ldots \wedge B_n$.*

Beweis. Mit struktureller Induktion über C. $\square$

Lemma 3.5.6 *Falls $B_1 \wedge \ldots \wedge B_n \;\approx\; B'_1 \wedge \ldots \wedge B'_m$, dann gilt $n = m$ und es existiert eine Bijektion $\theta : \{1, \ldots, n\} \to \{1, \ldots, n\}$ mit $B_{\theta(i)} \approx B'_i$ für alle $1 \leq i \leq n$.*

Beweis. Mit Induktion über Herleitungen von $B_1 \wedge \ldots \wedge B_n \;\approx_1\; B'_1 \wedge \ldots \wedge B'_m$ läßt sich die Behauptung für $\approx_1$ anstelle von $\approx$ zeigen und dies ist nach Lemma 3.5.2 hinreichend. Im Induktionsbeweis ist die Verwendung von $\approx_1$ anstelle von $\approx$ wesentlich für den Fall der Transitivität. $\square$

Lemma 3.5.7 *$C_1 \wedge C \approx C_2 \wedge C$ ist äquivalent zu $C_1 \approx C_2$.*

Beweis. Die Implikation von rechts nach links folgt aus Lemma 3.5.5 angewendet auf C und der zweiten Regel in der Definition von $\approx$. Für die Implikation von links nach rechts reicht es wegen Lemma 3.5.5 zu zeigen, daß aus $B_1 \wedge \ldots \wedge B_{n-1} \wedge B_n \approx B'_1 \wedge \ldots \wedge B'_{m-1} \wedge B'_m$ und $B_n \approx B'_m$ die Beziehung $B_1 \wedge \ldots \wedge B_{n-1} \approx B'_1 \wedge \ldots \wedge B'_{m-1}$ folgt. Wir nehmen also $B_1 \wedge \ldots \wedge B_{n-1} \wedge B_n \approx B'_1 \wedge \ldots \wedge B'_{m-1} \wedge B'_m$ und $B_n \approx B'_m$ an. Nach Lemma 3.5.6 gilt $n = m$ und es gibt eine Bijektion $\theta : \{1, \ldots, n\} \to \{1, \ldots, n\}$ mit $B_{\theta(i)} \approx B'_i$. Sei τ diejenige Bijektion auf $\{1, \ldots, n\}$, die n und $\theta(n)$ vertauscht und alle anderen Werte unverändert läßt. Dann ist $\theta' = \theta \circ \tau$ (zuerst θ, dann τ) eine Bijektion mit $\theta'(n) = n$ und mit $B_{\theta'(i)} \approx B'_i$. Insbesondere ist die Einschränkung von θ' auf $\{1, \ldots, n\}$ eine Bijektion und somit folgt:

$$B_1 \wedge \ldots \wedge B_{n-1} \approx B'_{\theta'(1)} \wedge \ldots \wedge B'_{\theta'(n-1)} \approx B'_1 \wedge \ldots \wedge B'_{n-1}$$

□

Letztendlich zeigen wir, daß sich Kongruenzklassen von Präfixen von Normalformen bezüglich $\approx$ wie Mengen verhalten.

Lemma 3.5.8 *Sind $\overline{x}$ und $\overline{y}$ linear, dann gilt $\exists\overline{x} \approx \exists\overline{y}$ genau dann, wenn $\mathcal{V}(\overline{x}) = \mathcal{V}(\overline{y})$.*

Beweis. Mit Induktion über Herleitungen von $\exists\overline{x}\top \equiv_{Exch} \exists\overline{y}\top$ bzw. mit Induktion über die Anzahl der Elemente von $\mathcal{V}(\overline{x})$. □

Substitutionen.

Wir wollen zunächst die Notwendigkeit von Nebenbedingungen über gebundene Variablen im Kontext von Substitution formal begründen.

Sei σ stets eine Substitution, d.h. eine Abbildung von Variablen auf Variablen. Dann definieren wir den Definitionsbereich $\mathcal{D}(\sigma)$ und den Bildbereich $\mathcal{R}(\sigma)$ durch:

$$\mathcal{D}(\sigma) = \{x \mid \sigma(x) \neq x\} \qquad \text{und} \qquad \mathcal{R}(\sigma) = \{\sigma(x) \mid x \in \mathcal{D}(\sigma)\}\,.$$

Die folgenden beiden Lemmata begründen die Nebenbedingungen an gebundene Variablen in (E), (A) und (α) formal:

Lemma 3.5.9 *$\mathcal{D}([\overline{y}/\overline{x}]) \subseteq \mathcal{V}(\overline{x})$ und $\mathcal{R}([\overline{y}/\overline{x}]) \subseteq \mathcal{V}(\overline{y})$.*

Beweis. Folgt direkt aus den Definitionen. □

Im folgenden schreiben wir $V\sigma$ für die elementweise Anwendung von σ auf eine Menge von Variablen V.

Lemma 3.5.10 *Sind $\mathcal{R}(\sigma)$ und $\mathcal{BV}(E)$ disjunkt, dann gilt $\mathcal{FV}(E\sigma) = \mathcal{FV}(E)\sigma$.*

Beweis. Mit struktureller Induktion über E. □

Nun notieren wir drei Gleichungen für Substitutionsoperatoren $[y/x]$ und $[\overline{y}/\overline{x}]$, die wir unter anderem für unsere Konfluenzbeweise häufig anwenden werden. Wir machen darauf aufmerksam, daß $\sigma \circ \sigma'$ auch hier für "σ vor σ'" steht.

Lemma 3.5.11 *Es gilt $[\overline{z}/\overline{y}] \circ [w'/w] = [w'/w] \circ [\overline{z}[w'/w]/\overline{y}[w'/w]]$, falls $w' \notin \mathcal{V}(\overline{y})$.*

Beweis. Beide Substitutionen bilden Variablen, die nicht in $\mathcal{V}(ww'\overline{y})$ liegen, auf sich selbst ab. Im folgenden bezeichnen wir mit σ stets die Substitution:

$$\sigma \;=\; [\overline{z}[w'/w]/\overline{y}[w'/w]]\,.$$

Wir nehmen ohne Beschränkung der Allgemeinheit $w \neq w'$ an, denn anderenfalls stimmen beide Substitutionen mit $[\overline{z}/\overline{y}]$ überein. Betrachten wir nun die Variable w':

$$\begin{array}{llll} w'[\overline{z}/\overline{y}] = w' & \text{und} & w'[w'/w] = w'\,, & \text{da } w' \notin \mathcal{V}(\overline{y}). \\ w'[w'/w] = w' & \text{und} & \sigma(w') = w'\,, & \text{denn wegen } w \neq w' \text{ gilt } w \notin \mathcal{V}(\overline{y}[w'/w]). \end{array}$$

Sei weiterhin $\overline{y} = (y_1, \ldots, y_n)$ und $\overline{z} = (z_1, \ldots, z_n)$. Betrachten wir nun ein beliebiges y_i:

$$\begin{array}{lll} y_i[\overline{z}/\overline{y}] = z_i & \text{und} & z_i[w'/w] = z_i[w'/w]\,, \\ y_i[w'/w] = y_i[w'/w] & \text{und} & \sigma(y_i[w'/w]) = z_i[w'/w]\,. \end{array}$$

Letztendlich kommen wir zur Variablen w. Wir können $w \notin \mathcal{V}(\overline{y})$ annehmen, da wir bereits wissen, daß beide betrachteten Substitutionen für alle y_i übereinstimmen. Dann gilt:

$$\begin{array}{llll} w[\overline{z}/\overline{y}] = w & \text{und} & w[w'/w] = w'\,, & \\ w[w'/w] = w' & \text{und} & \sigma(w') = w'\,, & \text{da } \overline{y}[w'/w] = \overline{y} \text{ und } w' \notin \mathcal{V}(\overline{y}). \end{array}$$ □

Lemma 3.5.12 *Es gilt $[\overline{z}/\overline{y}] \circ [w'/w] = [w'/w] \circ [\overline{z}[w'/w]/\overline{y}]$, falls $\mathcal{V}(\overline{y})$ und $\{w', w\}$ disjunkt sind.*

Beweis. Folgt unmittelbar aus Lemma 3.5.11, wegen $\overline{y}[w'/w] = \overline{y}$. □

Lemma 3.5.13 *Aus* $y \notin \{w', w\}$ *und* ($z \neq w$ *oder* $z = w'$) *folgt* $[z/y] \circ [w'/w] = [w'/w] \circ [z/y]$.

Beweis. Durch Einsetzen in Lemma 3.5.12. □

Wir werden im Konfluenzbeweis zwei Eigenschaften über Substitutionsinvarianz benötigen.

Definition 3.5.14 *Eine binäre Relation* R *auf* Δ *heißt* invariant unter Substitution, *falls für alle* $(E, F) \in R$ *und* $y \notin \mathcal{BV}(E) \cup \mathcal{BV}(F)$ *auch* $(E[y/x], F[y/x]) \in R$ *gilt.*

Diesen Begriff werden wir in Abschnitt 5.2.4 erneut benötigen und dann ausführlicher untersuchen.

Lemma 3.5.15 *Eine Relation* R *ist invariant unter Substitution, wenn für alle* $(E, F) \in R$ *auch* $(E[\overline{y}/\overline{x}], F[\overline{y}/\overline{x}]) \in R$ *gilt, falls* $\mathcal{V}(\overline{y})$ *und* $\mathcal{BV}(E) \cup \mathcal{BV}(F)$ *disjunkt sind.*

Beweis. Jeder Operator $[\overline{z}/\overline{y}]$ läßt sich als endliche Komposition von Substitionen $[z/y]$ darstellen. Ist z eine frische Variable, dann gilt zum Beispiel $[(y, x)/(x, y)] = [z/x] \circ [x/y] \circ [y/x]$ □

Lemma 3.5.16 *Die Relation* $\approx$ *auf chemischen Lösungen ist invariant unter Substitution.*

Beweis. Mit Induktion über Ableitungen von $C_1 \approx C_2$. Dazu benötigen wir die Invarianz von $\equiv_2$ unter Substitution, die in Proposition 5.2.13 gezeigt wird und sich auch an dieser Stelle ohne zusätzliche Schwierigkeiten zeigen ließe. □

3.5.2 Kritische Paare

Wir beweisen nun die Vertauschungsaussagen von Satz 3.1.7 und Satz 3.3.3. Dabei verwenden wir Definitionsbasen von Satz 3.4.4, was durch die Propositionen 2.8.6, 2.8.7 und 2.8.8 gerechtfertigt wird. Dank dieser Vorbereitungen können wir uns hier auf den paarweisen Vergleich von Applikation (A), Elimination (E) und Annullierung (G) zurückziehen.

Lemma 3.5.17 (Applikation-Applikation) *Seien D_1, D_2 zulässige Normalformen und G_1, G_2 Ausdrücke mit $G_1 \ {}_{A_t}\!\overleftarrow{\ } \ D_1 \equiv_2 D_2 \ \overrightarrow{\ }_{A_t} \ G_2$. Dann gilt entweder $G_1 \equiv G_2$ oder es existiert G mit $G_1 \to_A G \ {}_A\!\leftarrow \ G_2$.*

Beweis. Nach Voraussetzung des Lemmas und der Definition von $\overrightarrow{\ }_{A_t}$ haben D_1, D_2, G_1 und G_2 die folgenden Formen:

$$\begin{array}{rcl} D_1 & = & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge E_1) \ \overrightarrow{\ }_{A_t} \ \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E_1) \ = \ G_1 \\ D_2 & = & \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge x_2\,\overline{z_2} \wedge E_2) \ \overrightarrow{\ }_{A_t} \ \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E_2) \ = \ G_2 \end{array}$$

Aus $D_1 \equiv_2 D_2$ und Proposition 3.5.1 folgt dann:

$$\exists\overline{u_1} \ \approx \ \exists\overline{u_2}, \qquad x_1{:}\overline{y_1}/F_1 \wedge x_1\overline{z_1} \wedge E_1 \ \approx \ x_2{:}\overline{y_2}/F_2 \wedge x_2\overline{z_2} \wedge E_2 \ .$$

Da D_1 und D_2 PNF's sind, sind E_1 und E_2 ebenfalls PNF's, sowie E_1 und E_2 chemische Lösungen.

1. **Fall** $x_1 = x_2$ **:** Aus der Zulässigkeit von D_1 folgt $x_1{:}\overline{y_1}/F_1 \approx x_2{:}\overline{y_2}/F_2$; denn anderenfalls ließe sich die Inkonsistenz von $x_1{:}\overline{y_1}/F_1 \wedge E_1$ mit Lemma 3.5.5 und Lemma 3.5.6 herleiten. Somit gilt nach Lemma 3.5.3 und Lemma 3.5.7:

   $$\overline{y_1} = \overline{y_2}, \qquad F_1 \equiv_2 F_2, \qquad x_1\overline{z_1} \wedge E_1 \ \approx \ x_1\overline{z_2} \wedge E_2 \ .$$

   Wir unterscheiden nun zwei Unterfälle:

   (a) Im Fall $\overline{z_1} = \overline{z_2}$ folgt $E_1 \approx E_2$ aus Lemma 3.5.7 und somit erhalten wir mit Proposition 3.5.1:

   $$\begin{array}{rcl} G_1 & = & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E_1) \\ & \equiv_2 & \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E_2) \\ & = & G_2 \end{array}$$

   (b) Im Fall $\overline{z_1} \neq \overline{z_2}$ gilt $x_1\,\overline{z_1} \not\approx x_1\,\overline{z_2}$ wegen Lemma 3.5.4 und somit folgt aus Lemma 3.5.6 und Lemma 3.5.5 für ein geeignetes E:

   $$E_1 \ \approx \ x_2\overline{z_2} \wedge E \ , \qquad E_2 \ \approx \ x_1\overline{z_1} \wedge E \ .$$

   Die Behauptung erhalten wir mit Proposition 3.5.1 wie folgt:

   $$\begin{array}{rcl} G_1 & = & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E_1) \\ & \equiv_2 & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge x_2\,\overline{z_2} \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \\ & \overrightarrow{\ }_A & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_2}/\overline{y_1}] \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \\ \\ G_2 & = & \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E_2) \\ & \equiv_2 & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge F_1[\overline{z_2}/\overline{y_1}] \wedge E) \\ & \overrightarrow{\ }_A & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge F_1[\overline{z_2}/\overline{y_1}] \wedge E) \\ & \equiv_2 & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_2}/\overline{y_1}] \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \end{array}$$

Die Nebenbedingungen für die Anwendungen von $\bar{\to}_A$ folgen aus α-Standardisierung.

2. **Fall** $x_1 \neq x_2$ **:** Für ein geeignetes E gilt:

$$E_1 \;\approx\; x_2{:}\overline{y_2}/F_2 \wedge x_2\,\overline{z_2} \wedge E\,, \qquad E_2 \;\approx\; x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge E\,.$$

Wir können die Behauptung wie folgt schließen:

$$\begin{array}{rl} G_1 \;=\; & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E_1) \\ \equiv_2 & \exists\overline{u_1}(x_2{:}\overline{y_2}/F_2 \wedge x_2\,\overline{z_2} \wedge x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \\ \bar{\to}_A & \exists\overline{u_1}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \\ \equiv_2 & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge x_2{:}\overline{y_2}/F_2) \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E) \end{array}$$

$$\begin{array}{rl} G_2 \;=\; & \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E_2) \\ \equiv_2 & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E) \\ \bar{\to}_A & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E) \end{array}$$

Wiederum gelten die Nebenbedingungen für die Anwendungen von $\bar{\to}_A$ wegen α-Standardisierung. □

Lemma 3.5.18 (Elimination-Elimination) *Seien D_1, D_2 Normalformen und G_1, G_2 Ausdrücke mit $G_1 \;{}_{E_t}\bar{\leftarrow}\; D_1 \equiv_2 D_2 \;\bar{\to}_{E_t}\; G_2$. Dann gilt entweder $G_1 \equiv G_2$ oder es existiert G mit $G_1 \to_E G \;{}_E\!\leftarrow\; G_2$.*

Beweis. Wir verzichten von nun an auf exakte Begründungen im Umgang mit $\equiv_2$ und $\approx$ und vertrauen auf die Intuition mit Multimengen. Nach Voraussetzung des Lemmas und Definition von (E_t) haben D_1, D_2, G_1 und G_2 die folgenden Formen:

$$\begin{array}{rcl} D_1 &=& \exists\overline{u_1}\exists x_1(x_1 = y_1 \wedge E_1) \;\bar{\to}_{E_t}\; G_1 \;=\; \exists\overline{u_1}E_1[y_1/x_1] \\ D_2 &=& \exists\overline{u_2}\exists x_2(x_2 = y_2 \wedge E_2) \;\bar{\to}_{E_t}\; G_2 \;=\; \exists\overline{u_2}E_2[y_2/x_2] \end{array}$$

Dabei gelten die Nebenbedingungen $y_1 \in \mathcal{V}(\overline{u_1})$ und $y_2 \in \mathcal{V}(\overline{u_2})$, so daß wegen α-Standardisierung $x_1 \neq y_1$ und $x_2 \neq y_2$ folgt. Die Voraussetzung $D_1 \equiv_2 D_2$ ist äquivalent zu:

$$\exists\overline{u_1}\exists x_1 \;\approx\; \exists\overline{u_2}\exists x_2\,, \qquad x_1 = y_1 \wedge E_1 \;\approx\; x_2 = y_2 \wedge E_2\,.$$

1. **Fall** $x_1 = x_2$ **:** Mit Lemma 3.5.8 und α-Standardisierung folgt $\exists\overline{u_1} \approx \exists\overline{u_2}$.

   (a) Im Fall $y_1 = y_2$ folgt $E_1 \approx E_2$. Nach Lemma 3.5.16 ist $\approx$ invariant unter Substitution, so daß $E_1[y_1/x_1] \approx E_2[y_1/x_1]$ gilt. Somit folgt:

$$G_1 \;=\; \exists\overline{u_1}E_1[y_1/x_1] \;\equiv_2\; \exists\overline{u_2}E_2[y_1/x_1] \;=\; \exists\overline{u_2}E_2[y_2/x_2] \;=\; G_2\,.$$

(b) Im Fall $y_1 \neq y_2$ folgt die Existenz von E mit

$$E_1 \;\approx\; x_1 = y_2 \wedge E\,, \qquad E_2 \;\approx\; x_1 = y_1 \wedge E\,.$$

Hier benötigen wir die Nebenbedingungen $y_1 \in \mathcal{V}(\overline{u_1})$ und $y_2 \in \mathcal{V}(\overline{u_2})$, die die Existenz von $\overline{u}$ mit

$$\exists\overline{u_1} \;\approx\; \exists\overline{u}\exists y_1\exists y_2 \;\approx\; \exists\overline{u_2}$$

sicherstellen, so daß nun die Behauptung unter Verwendung von α-Konversion (!!) folgt:

$$\begin{array}{rcll}
G_1 = \exists\overline{u_1}E_1[y_1/x_1] & \equiv_2 & \exists\overline{u}\exists y_2\exists y_1(y_1 = y_2 \wedge E[y_1/x_1]) \\
& \xrightarrow{-}_E & \exists\overline{u}\exists y_2(E[y_1/x_1][y_2/y_1]) \\
& = & \exists\overline{u}\exists y_2(E[y_2/x_1][y_2/y_1]) \\
G_2 = \exists\overline{u_2}E_2[y_2/x_2] & \equiv_2 & \exists\overline{u}\exists y_1\exists y_2(y_2 = y_1 \wedge E[y_2/x_2]) \\
& \xrightarrow{-}_E & \exists\overline{u}\exists y_1(E[y_2/x_1][y_1/y_2]) \\
& \equiv_\alpha & \exists\overline{u}\exists y_2(E[y_2/x_1][y_1/y_2][y_2/y_1]) \\
& = & \exists\overline{u}\exists y_2(E[y_2/x_1][y_2/y_1])
\end{array}$$

Dieser Fall wäre nicht durchgegangen, wenn die bei Elimination beteiligten Variablen y_1 und y_2 global sein dürften.

2. **Fall** $x_1 \neq x_2$ **:** Wir erhalten:

$$\begin{array}{rclcrcl}
\exists\overline{u_1} & \approx & \exists\overline{u}\exists x_2\,, & \qquad & \exists\overline{u_2} & \approx & \exists\overline{u}\exists x_1\,, \\
E_1 & \approx & x_2 = y_2 \wedge E\,, & & E_2 & \approx & x_1 = y_1 \wedge E\,.
\end{array}$$

(a) Im Fall $x_1 \neq y_2$ und $x_2 \neq y_1$ gilt:

$$\begin{array}{rcll}
G_1 = \exists\overline{u_1}\,E_1[y_1/x_1] & \equiv_2 & \exists\overline{u}\exists x_2(x_2 = y_2 \wedge E[y_1/x_1]) \\
& \xrightarrow{-}_E & \exists\overline{u}(E[y_1/x_1][y_2/x_2]) \\
G_2 = \exists\overline{u_2}E_2[y_2/x_2] & \equiv_2 & \exists\overline{u}\exists x_1(x_1 = y_1 \wedge E[y_2/x_2])) \\
& \xrightarrow{-}_E & \exists\overline{u}(E[y_2/x_2][y_1/x_1]) \\
& = & \exists\overline{u}(E[y_1/x_1][y_2/x_2]) & \text{Lemma 3.5.13}
\end{array}$$

(b) Im Fall $x_1 = y_2$ und $x_2 \neq y_1$ erhalten wir:

$$\begin{array}{rcll}
G_1 = \exists\overline{u_1}E_1[y_1/x_1] & \equiv_2 & \exists\overline{u}\exists x_2(x_2 = y_1 \wedge E[y_1/x_1]) \\
& \xrightarrow{-}_E & \exists\overline{u}(E[y_1/x_1][y_1/x_2]) \\
G_2 = \exists\overline{u_2}E_2[y_2/x_2] & \equiv_2 & \exists\overline{u}\exists x_1(x_1 = y_1 \wedge E[x_1/x_2]) \\
& \xrightarrow{-}_E & \exists\overline{u}(E[x_1/x_2][y_1/x_1]) \\
& = & \exists\overline{u}(E[y_1/x_2][y_1/x_1]) \\
& = & \exists\overline{u}(E[y_1/x_1][y_1/x_2]) & \text{Lemma 3.5.13}
\end{array}$$

(c) Der Fall $x_1 \neq y_2$ und $x_2 = y_1$ ist symmetrisch zum vorangehenden.

(d) Für den Fall $x_1 = y_2$ und $x_2 = y_1$ verwenden wir eine frische Variable z:

$$\begin{array}{rcll} G_1 & = & \exists\overline{u_1}\,E_1[y_1/x_1] & \equiv_2 \; \exists\overline{u}\exists y_1(y_1 = y_1 \wedge E[y_1/x_1]) \\ & & & \equiv_\alpha \; \exists\overline{u}\exists z(z = z \wedge E[y_1/x_1][z/y_1]) \\ & & & = \; \exists\overline{u}\exists z(z = z \wedge E[z/x_1][z/y_1]) \\ G_2 & = & \exists\overline{u_2}\,E_2[y_2/x_2] & \equiv_2 \; \exists\overline{u}\exists x_1(x_1 = x_1 \wedge E[x_1/y_1]) \\ & & & \equiv_\alpha \; \exists\overline{u}\exists z(z = z \wedge E[x_1/y_1][z/x_1]) \\ & & & = \; \exists\overline{u}\exists z(z = z \wedge E[z/y_1][z/x_1]) \\ & & & = \; \exists\overline{u}\exists z(z = z \wedge E[z/x_1][z/y_1]) \end{array}$$

Beide Gleichungen zwischen Substitutionen, die wir verwendet haben, sind durch Lemma 3.5.13 gerechtfertigt.

□

Lemma 3.5.19 (Applikation-Elimination) *Seien D_1, D_2 Normalformen und G_1, G_2 Ausdrücke mit $G_1 \;{}_{A_t}\!\!\leftarrow\; D_1 \equiv_2 D_2 \;\xrightarrow[E_t]{}\; G_2$. Dann existiert G mit $G_1 \;\rightarrow_E\; G \;{}_{A}\!\!\leftarrow\; G_2$.*

Beweis. Nach Voraussetzung des Lemmas und den Definitionen von (A_t) und (E_t) haben D_1, D_2, G_1 und G_2 die Bauart:

$$\begin{array}{rcl} D_1 & = & \exists\overline{u_1}\exists x_1(x_1 = y_1 \wedge E_1) \;\xrightarrow[E_t]{}\; \exists\overline{u_1}\,E_1[y_1/x_1] \;=\; G_1 \\ D_2 & = & \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge x_2\,\overline{z_2} \wedge E_2) \;\xrightarrow[A_t]{}\; \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E_2) \;=\; G_2 \end{array}$$

Dabei gilt $y_1 \in \mathcal{V}(\overline{u_1}) \cup \{x_1\}$. Die Voraussetzung $D_1 \equiv_2 D_2$ impliziert $\exists\overline{u_1}\exists x_1 \approx \exists\overline{u_2}$, sowie die Existenz eines E mit:

$$E_2 \;\approx\; x_1 = y_1 \wedge E\,, \qquad E_1 \;\approx\; x_2{:}\overline{y_2}/F_2 \wedge x_2\overline{z_2} \wedge E\,.$$

Die Existenz von G folgt nun aus den folgenden beiden Rechnungen:

$$\begin{array}{rcl} G_1 & = & \exists\overline{u_1}\,E_1[y_1/x_1] \\ & \equiv_2 & \exists\overline{u_1}(x_2[y_1/x_1]{:}\overline{y_2}/F_2[y_1/x_1] \wedge x_2[y_1/x_1]\,\overline{z_2}[y_1/x_1] \wedge E[y_1/x_1]) \\ & \xrightarrow{}_A & \exists\overline{u_1}(x_2[y_1/x_1]{:}\overline{y_2}/F_2[y_1/x_1] \wedge F_2[y_1/x_1][(\overline{z_2}[y_1/x_1])/\overline{y_2}] \wedge E[y_1/x_1]) \\ G_2 & = & \exists\overline{u_2}(x_2{:}\overline{y_2}/F_2 \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E_2) \\ & \equiv_2 & \exists\overline{u_1}\exists x_1(x_1 = y_1 \wedge x_2{:}\overline{y_2}/F_2) \wedge F_2[\overline{z_2}/\overline{y_2}] \wedge E) \\ & \xrightarrow{}_E & \exists\overline{u_1}(x_2[y_1/x_1]{:}\overline{y_2}/F_2[y_1/x_1] \wedge F_2[\overline{z_2}/\overline{y_2}][y_1/x_1] \wedge E[y_1/x_1]) \end{array}$$

Die Ergebnisse von beiden Rechnungen stimmen überein; denn wegen Lemma 3.5.12 gilt:

$$[\overline{z_2}/\overline{y_2}] \circ [y_1/x_1] \;=\; [y_1/x_1] \circ [(\overline{z_2}[y_1/x_1])/\overline{y_2}]\,.$$

Die Voraussetzung für dessen Anwendung, nämlich $\mathcal{V}(\overline{y})$ disjunkt zu $\{y_1, x_1\}$, folgt aus α-Standardisierung. □

Lemma 3.5.20 (Elimination-Annullierung) *Seien D_1, D_2 Normalformen und G_1, G_2 Ausdrücke mit $G_1 \ {}_{E_t}\!\leftarrow \ D_1 \equiv_2 D_2 \ \bar{\rightarrow}_{G_t} \ G_2$. Dann existiert G mit $G_1 \ \rightarrow_G \ G \ {}_{E}\!\leftarrow \ G_2$.*

Beweis. Aus der Voraussetzung des Lemmas und den Definitionen von (E_t) und (G_t) folgt, daß D_1, D_2, G_1 und G_2 folgende Struktur haben:

$$\begin{array}{rcl} D_1 & = & \exists\overline{u_1}\exists x_1(x_1 = y_1 \wedge E_1) \ \bar{\rightarrow}_{E_t} \ \exists\overline{u_1}E_1[y_1/x_1] \ = \ G_1 \\ D_2 & = & \exists\overline{u_2}\exists\overline{x_2}(\overline{y_2}\colon \overline{A_2} \wedge E_2) \ \bar{\rightarrow}_{G_t} \ \exists\overline{u_2}(E_2) \ = \ G_2 \end{array}$$

Wegen der Nebenbedingungen der Axiome können wir desweiteren

$$y_1 \in \mathcal{V}(\overline{u_1})\,, \qquad \mathcal{V}(\overline{y_2}) \subseteq \mathcal{V}(\overline{x_2})\,, \qquad \mathcal{V}(\overline{x_2}) \cap \mathcal{FV}(E_2) = \emptyset$$

annehmen. Aus α-Standardisierung folgt $x_1 \neq y_1$ und wegen $D_1 \equiv_2 D_2$ existiert E mit:

$$\exists\overline{u_1}\exists x_1 \ \approx \ \exists\overline{u_2}\exists\overline{x_2}\,, \qquad E_1 \ \approx \ \overline{y_2}\colon \overline{A_2} \wedge E\,, \qquad E_2 \ \approx \ x_1 = y_1 \wedge E\,.$$

Daraus erhalten wir $\{x_1, y_1\} \subseteq \mathcal{FV}(E_2)$, $\{x_1, y_1\} \cap \mathcal{V}(\overline{x_2}) = \emptyset$ und $\{x_1, y_1\} \subseteq \mathcal{V}(\overline{u_2})$. Desweiteren existieren $\overline{v_1}$ und $\overline{v_2}$ mit $\exists\overline{u_1} \approx \exists\overline{v_1}\exists y_1$ und $\exists\overline{u_2} \approx \exists\overline{v_2}\exists x_1\exists y_1$. Dann gilt auch $\exists\overline{v_1} \ \approx \ \exists\overline{v_2}\exists\overline{x_2}$.

$$\begin{array}{rcll} G_1 = \exists\overline{u_1}E_1[y_1/x_1] & \equiv_2 & \exists\overline{v_2}\exists y_1\exists\overline{x_2}(\overline{y_2}\colon \overline{A_2}[y_1/x_1] \wedge E[y_1/x_1]) \\ & \bar{\rightarrow}_G & \exists\overline{v_2}\exists y_1 E[y_1/x_1] \\ G_2 = \exists\overline{u_2}(E_2) & \equiv_2 & \exists\overline{v_2}\exists x_1\exists y_1(x_1 = y_1 \wedge E) \\ & \bar{\rightarrow}_E & \exists v_2\exists y_1 E[y_1/x_1] \end{array}$$

Um die Anwendbarkeit von $\bar{\rightarrow}_G$ sicherzustellen, müssen wir $\mathcal{V}(\overline{x_2}) \cap \mathcal{FV}(E[y_1/x_1]) = \emptyset$ nachweisen. Wegen der Lemmata 3.5.10 und 3.5.9 gilt:

$$\mathcal{FV}(E[y_1/x_1]) \ = \ \mathcal{FV}(E)[y_1/x_1] \ \subseteq \ \mathcal{FV}(E) \cup \{y_1\} \ \subseteq \ \mathcal{FV}(E_2)\,.$$

Daraus folgt die gesuchte Bedingung, denn $\mathcal{V}(\overline{x_2}) \cap \mathcal{F}(E_2) = \emptyset$ ist vorausgesetzt. □

Lemma 3.5.21 (Applikation-Annullierung) *Seien D_1, D_2 Normalformen und G_1, G_2 Ausdrücke mit $G_1 \ {}_{A_t}\!\leftarrow \ D_1 \equiv_2 D_2 \ \bar{\rightarrow}_{G_t} \ G_2$. Dann existiert G mit $G_1 \ \rightarrow_G \ G \ {}_{A}\!\leftarrow \ G_2$.*

Beweis. Aus der Voraussetzung des Lemmas und den Definitionen von (A_t) und (G_t) folgt, daß D_1, D_2, G_1 und G_2 folgendermaßen aussehen:

$$\begin{array}{rcl} D_1 & = & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge E_1) \;\bar{\to}_{U_t}\; \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E_1) \;=\; G_1 \\ D_1 & = & \exists\overline{u_2}\exists\overline{x_2}(\overline{y_2}{:}\,\overline{A_2} \wedge E_2) \;\bar{\to}_{G_t}\; \exists\overline{u_2}E_2 \;=\; G_2 \end{array}$$

Desweiteren setzt die Annullierung folgende Nebenbedingungen voraus:

$$\mathcal{V}(\overline{y_2}) \subseteq \mathcal{V}(\overline{x_2})\,, \qquad \mathcal{V}(\overline{x_2}) \cap \mathcal{FV}(E_2) = \emptyset\,.$$

Aus der Voraussetzung $D_1 \equiv_2 D_2$ erhalten wir:

$$\exists\overline{u_1} \;\approx\; \exists\overline{u_2}\exists\overline{x_2}\,, \qquad x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge E_1 \;\approx\; \overline{y_2}{:}\,\overline{A_2} \wedge E_2\,.$$

Aus den Nebenbedingungen können wir nun die Existenz von E schließen, welches

$$E_2 \;\approx\; x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge E\,, \qquad E_1 \;\approx\; \overline{y_2}{:}\,\overline{A_2} \wedge E$$

erfüllt. Das Lemma folgt nun aus den folgenden beiden Rechnungen:

$$\begin{array}{rcl} G_1 & = & \exists\overline{u_1}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E_1) \\ & \equiv_2 & \exists\overline{u_2}\exists\overline{x_2}(\overline{y_2}{:}\,\overline{A_2} \wedge x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \\ & \bar{\to}_G & \exists\overline{u_2}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \end{array}$$

$$\begin{array}{rcl} G_2 & = & \exists\overline{u_2}E_2 \\ & \equiv_2 & \exists\overline{u_2}(x_1{:}\overline{y_1}/F_1 \wedge x_1\,\overline{z_1} \wedge E) \\ & \bar{\to}_A & \exists\overline{u_2}(x_1{:}\overline{y_1}/F_1 \wedge F_1[\overline{z_1}/\overline{y_1}] \wedge E) \end{array}$$

Die Voraussetzung für die Anwendbarkeit von $\to_G$ läßt sich wiederum leicht nachrechnen. □

Lemma 3.5.22 (Annullierung-Annullierung) *Seien D_1, D_2 Normalformen und G_1, G_2 Ausdrücke mit $G_1 \;{}_{G_t}\bar{\leftarrow}\; D_1 \equiv_2 D_2 \;\bar{\to}_{G_t}\; G_2$. Dann existiert G mit $G_1 \to^{\epsilon}_{G} G \;{}^{\epsilon}_{G}\!\leftarrow G_2$.*

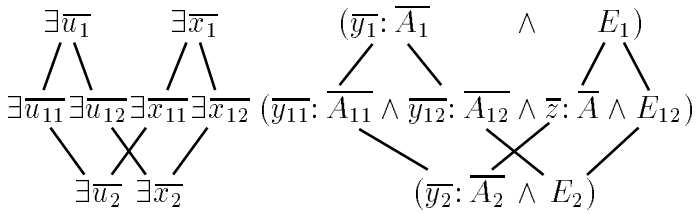
Beweis. Aus der Voraussetzung des Lemmas und den Definitionen von (E_t) und (G_t) folgt, daß D_1, D_2, G_1 und G_2 folgende Struktur haben:

$$\begin{array}{rcl} D_1 & = & \exists\overline{u_1}\exists\overline{x_1}(\overline{y_1}{:}\,\overline{A_1} \wedge E_1) \;\bar{\to}_{G_t}\; \exists\overline{u_1}E_1 \;=\; G_1 \\ D_2 & = & \exists\overline{u_2}\exists\overline{x_2}(\overline{y_2}{:}\,\overline{A_2} \wedge E_2) \;\bar{\to}_{G_t}\; \exists\overline{u_2}E_2 \;=\; G_2 \end{array}$$

Wir werden im folgenden mehrfach die Disjunktheit von Mengen K und L benötigen und schreiben dann abkürzend $K \parallel L$ anstelle von $K \cap L = \emptyset$. Aus den Nebenbedingungen für die obigen Reduktionen erhalten wir:

$$\begin{array}{rcl}]\mathcal{V}(\overline{y_1}) & \subseteq & \mathcal{V}(\overline{x_1})\,, \qquad \mathcal{V}(\overline{x_1}) \;\parallel\; \mathcal{FV}(E_1)\,, \\ \mathcal{V}(\overline{y_2}) & \subseteq & \mathcal{V}(\overline{x_2})\,, \qquad \mathcal{V}(\overline{x_2}) \;\parallel\; \mathcal{FV}(E_2)\,. \end{array}$$

Aus der Voraussetzung $D_1 \equiv_2 D_2$ folgt die Existenz einer ganzen Sammlung von Präfixen und Ausdrücken, sowie eine Sammlung von $\approx$ Beziehungen, die wir durch folgende Abbildung darstellen:



Zum Beispiel gilt $\exists\overline{u_1} \approx \exists\overline{u_{11}}\exists\overline{u_{12}}$ und $\exists\overline{u_2} \approx \exists\overline{u_{12}}\exists\overline{x_{11}}$. Dies erlaubt uns wie folgt zu reduzieren:

$$\begin{array}{rcl} G_1 & = & \exists\overline{u_1}E_1 \equiv_2 \exists\overline{u_{11}}\exists\overline{u_{12}}(\overline{z}\colon \overline{A} \wedge E_{12}) \stackrel{-}{\rightarrow}^{\epsilon}_{G} \exists\overline{u_{11}}E_{12} \\ G_2 & = & \exists\overline{u_2}E_2 \equiv_2 \exists\overline{u_{11}}\exists\overline{x_{11}}(\overline{y_{12}}\colon \overline{A_{12}} \wedge E_{12}) \stackrel{-}{\rightarrow}^{\epsilon}_{G} \exists\overline{u_{11}}E_{12} \end{array}$$

Um die beiden $\stackrel{-}{\rightarrow}^{\epsilon}_{G}$ zu rechtfertigen, müssen wir die folgenden Bedingungen nachrechnen:

$$\begin{array}{rcl@{\qquad}rcl} \mathcal{V}(\overline{z}) & \subseteq & \mathcal{V}(\overline{u_{12}})\,, & \mathcal{V}(\overline{u_{12}}) & \parallel & \mathcal{FV}(E_{12})\,, \\ \mathcal{V}(\overline{y_{12}}) & \subseteq & \mathcal{V}(\overline{x_{11}})\,, & \mathcal{V}(\overline{x_{11}}) & \parallel & \mathcal{FV}(E_{12})\,. \end{array}$$

Wir zeigen zuerst $\mathcal{V}(\overline{z}) \subseteq \mathcal{V}(\overline{u_{12}})$. Dies können wir aus den folgenden Abschätzungen schließen:

$$\begin{array}{rcl@{\quad}rcl} \mathcal{V}(\overline{z}) & \subseteq & \mathcal{FV}(E_1) & \parallel & \mathcal{V}(\overline{x_1}) & = & \mathcal{V}(\overline{x_{11}}) \cup \mathcal{V}(\overline{x_{12}}) \\ \mathcal{V}(\overline{z}) & \subseteq & \mathcal{V}(\overline{y_2}) & \subseteq & \mathcal{V}(\overline{x_2}) & = & \mathcal{V}(\overline{u_{12}}) \cup \mathcal{V}(\overline{x_{12}}) \end{array}$$

Die Bedingung $\mathcal{V}(\overline{u_{12}}) \parallel \mathcal{FV}(E_{12})$ erhalten wir nun aus:

$$\mathcal{FV}(E_{12}) \subseteq \mathcal{FV}(E_2) \parallel \mathcal{V}(\overline{x_2}) = \mathcal{V}(\overline{u_{12}}) \cup \mathcal{V}(\overline{x_{12}})\,.$$

Die Beweise von $\mathcal{V}(\overline{y_{12}}) \subseteq \mathcal{V}(\overline{x_{11}})$ und $\mathcal{V}(\overline{x_{11}}) \parallel \mathcal{FV}(E_{12})$ sind symmetrisch:

$$\begin{array}{rcl@{\quad}rcl} \mathcal{V}(\overline{y_{12}}) & \subseteq & \mathcal{FV}(E_2) & \parallel & \mathcal{V}(\overline{x_2}) & = & \mathcal{V}(\overline{u_{12}}) \cup \mathcal{V}(\overline{x_{12}}) \\ \mathcal{V}(\overline{y_{12}}) & \subseteq & \mathcal{V}(\overline{y_1}) & \subseteq & \mathcal{V}(\overline{x_1}) & = & \mathcal{V}(\overline{x_{11}}) \cup \mathcal{V}(\overline{x_{12}}) \\ \mathcal{FV}(E_{12}) & \subseteq & \mathcal{FV}(E_1) & \parallel & \mathcal{V}(\overline{x_1}) & = & \mathcal{V}(\overline{x_{11}}) \cup \mathcal{V}(\overline{x_{12}}) \end{array}$$

□

Kapitel 4

Funktionale Berechnung

Strikte funktionale Berechnung wird üblicherweise durch den strikten λ-Kalkül modelliert und nicht-strikte funktionale Berechnung durch den nicht-strikten λ-Kalkül [Win93]. Wir wiederholen die Definition dieser beiden Kalkülen, sowie eine wohlbekannte Einbettung des nicht-strikten in den strikten λ-Kalkül [DH92, Plo75]. Weiterhin betten wir beide λ-Kalküle in den δ-Kalkül ein, wobei wir [Smo94c] folgen. Die originären Beiträge dieses Kapitels sind von technischer Natur, aber keineswegs langweilig.

- Wir zeigen, daß die aus den Einbettungen der beiden λ-Kalküle stammenden Ausdrücke von Δ zulässig sind, wodurch wir Inkonsistenzen bei Reduktion von diesen Ausdrücken statisch ausschließen.
- Wir beweisen die Adäquatheit der Einbettung des strikten λ-Kalküls in den δ-Kalkül bezüglich Komplexität und Terminierung. Dazu kombinieren wir Robin Milners Technik für explizite Substitutionen [Mil92] mit uniformer Konfluenz (Satz 2.7.4)).

Den Beweis für die Adäquatheit der Einbettung des nicht-strikten λ-Kalküls in den δ-Kalkül bleiben wir hier schuldig. Zum Zeitpunkt der Abgabe der Arbeit war dem Autor jedoch ein Beweis bekannt. Dieser kombiniert wiederum die Werkzeuge aus Kapitel 2 mit Robin Milner's Technik für explizite Substitutionen und verwendet desweiteren eine Idee, die unabhängig für die Beweise in [AFM$^+$95] benötigt und gefunden wurde.

4.1 Strikte funktionale Berechnung

Wir beginnen mit der Definition des strikten λ-Kalküls ("eager λ-calculus"), der ein geeignetes Komplexitätsmodell für strikte funktionale Berechnung im Stile von ML

[HMM86], Lisp [WH81] und Scheme [CR91] ist. λ-Ausdrücke M, N sind Variablen, Abstraktionen oder Applikationen, deren abstrakte Syntax wohlbekannt ist.

$$\begin{array}{rcl} V, W & ::= & \lambda x.M \\ M, N & ::= & x \mid V \mid MN \end{array}$$

Eine Abstaktion $\lambda x.M$ besteht aus einem formalen Argument x und Rumpf M und eine Applikation MN aus Funktion M und Argument N. Die Menge aller λ-Ausdrücke bezeichnen wir mit Λ.

Gebundene Variablen werden als formale Argumente von Abstraktionen eingeführt. Die Menge aller freien bzw. gebundenen Variablen in M denotieren wir mit $\mathcal{FV}(M)$ und $\mathcal{BV}(M)$. Ein λ-Ausdruck heißt geschlossen ("closed"), falls er keine freien Variablen enthält. Nur geschlossene λ-Ausdrücke entsprechen funktionalen Programmen. Die Menge aller geschlossenen λ-Ausdrücke bezeichnen wir mit Λ^{cl}.

Eine Kongruenz auf λ-Ausdrücken ist eine Äquivalenzrelation, die in beliebigen Kontexten angewendet werden darf. Wir definieren die strukturelle Kongruenz $\equiv$ als kleinste Kongruenz auf λ-Ausdrücken, die konsistentes Umbenennen gebundener Variablen ermöglicht.

Im Kontext des λ-Kalkül verstehen wir unter einer Substitution eine Abbildung von Variablen auf Terme. Der Operator $[N/x]$ steht für diejenige Substitution, die x auf N abbildet und alle anderen Variablen auf sich selbst. Wie üblich schreiben wir $M[N/x]$ für die Anwendung dieses Operators auf M.

Die Reduktion $\rightarrow_e$ des strikten λ-Kalküls ist die kleinste binäre Relation auf Λ, die die Axiome und Regeln in Abbildung 4.1 erfüllt.

$$(\beta_e) \quad (\lambda x.M)V \rightarrow_e M[V/x] \quad \text{falls } \mathcal{FV}(V) \cap \mathcal{BV}(M) = \emptyset$$

$$(Func) \quad \frac{M \rightarrow_e M'}{MN \rightarrow_e M'N} \qquad (Arg) \quad \frac{N \rightarrow_e N'}{MN \rightarrow_e MN'}$$

$$(Congr) \quad \frac{M_1 \equiv M_2 \qquad M_2 \rightarrow_e N_2 \qquad N_2 \equiv N_1}{M_1 \rightarrow_e N_1}$$

Abbildung 4.1: Reduktion des strikten λ-Kalküls

Als strikten λ-Kalkül bezeichnen wir das Tripel $\lambda_e = (\Lambda, \equiv, \rightarrow_e)$.

Reduktion in Abstraktionen ist nicht zugelassen. Es ist leicht zu zeigen, daß Abstraktionen die einzigen bezüglich $\rightarrow_e$ irreduziblen geschlossenen λ-Ausdrücke sind. Eine strikte Berechnung eines λ-Ausdruckes terminiert also mit einer Abstraktion oder überhaupt nicht.

Die Reduktion des strikten λ-Kalküls $\rightarrow_e$ verwendet nur eine eingeschränkte Form des Operators $[N/x]$, nämlich $[V/x]$ in (β_e). Dieser ist harmlos im Hinblick auf Komplexität. Denn Abstraktionen sind irreduzibel, so daß die Anwendung von $[V/x]$ auf eine beliebigen Ausdruck M keine Berechnung verdoppeln kann. Funktionale Argumente werden im strikten λ-Kalkül höchstens einmal ausgewertet, da sie vor Applikation berechnet sein müssen. Steht zum Beispiel I für die λ-Identität $\lambda x.x$ und $Copy$ für $\lambda x.xx$, so wird die Berechnung von II bei Reduktion von $Copy\ (II)$ geteilt:

$$Copy\ (II)\ \rightarrow_e\ Copy\ I\ \rightarrow_e\ II\ \rightarrow_e\ I\ .$$

Die Konfluenz des strikten λ-Kalküls und die Tatsache, daß Terminierung im strikten λ-Kalkül unabhängig von der Berechnungsreihenfolge ist, sind wohlbekannt und lassen sich mit Hilfe von uniformer Konfluenz beweisen.

Satz 4.1.1 *Der strikte λ-Kalkül ist uniform konfluent.*

Beweis. Mit Induktion über die Struktur von λ-Ausdrücken, (die durch α-Umbenennung erhalten bleibt). Da Variablen und Abstraktionen irreduzibel sind, reicht es aus, verschiedene Reduktionsschritte auf Applikationen zusammenzuführen. Wir betrachte dazu die Situation $\tilde{M}_1 \leftarrow MN \rightarrow \tilde{M}_2$. Läßt sich (β_e) direkt auf MN oder eine kongruente Variante anwenden, dann sind M und N Abstraktionen und somit irreduzibel. In diesem Fall sind beide Reduktionsschritte bis auf α-Umbenennung identisch und es gilt $\tilde{M}_1 \equiv \tilde{M}_2$.

Werden beide $\rightarrow_e$-Schritte innerhalb von der Funktion M ausgeführt, folgt die Behauptung aus der Induktionsvoraussetzung und Regel $(Func)$. Reduzieren beide $\rightarrow_e$-Schritte das Argument N, so folgt die Behauptung aus der Induktionsvoraussetzung und Regel (Arg). Es verbleibt der Fall, daß jeweils ein $\rightarrow_e$-Schritt in M und einer in N lokalisiert ist. Dann gilt entwender $M \rightarrow_e M'$ und $N \rightarrow_e N'$, $\tilde{M}_1 \equiv M'N$ und $\tilde{M}_2 \equiv MN'$ oder der symmetrische Fall, in dem $\tilde{M}_1$ und $\tilde{M}_2$ die Rollen tauschen. Die Behauptung folgt nun wegen $\tilde{M}_1 \equiv M'N \rightarrow M'N' \leftarrow MN' \equiv \tilde{M}_2$. □

4.1.1 Der strikte λ-Kalkül im δ-Kalkül

Wir betten nun den strikten λ-Kalkül in den δ-Kalkül ein, so daß Komplexität für geschlossene λ-Ausdrücke erhalten bleibt. Die Einbettung beruht im wesentlichen in der Benennung aller Unterausdrücke eines λ-Ausdruckes, wobei funktionale Schachtelung durch Deklaration und Komposition aufgelöst wird.

Formal definieren wir benannte λ-Ausdrücke als Paare bestehend aus einer Variablen x und einem λ-Ausdruck N und denotieren solche Paare durch $x = N$. Wir machen darauf aufmerksam, daß $x = y$ sowohl für einen benannten λ-Ausdruck, als auch für

einen δ-Ausdruck steht. In Abbildung 4.2 definieren wir eine Relation $\equiv$ zwischen benannten λ-Ausdrücken und Δ.

$$\frac{z = M \;\equiv\; E}{x = \lambda y.M \;\equiv\; x{:}y\,z/E} \qquad\qquad x = y \;\equiv\; x = y$$

$$\frac{y = M \;\equiv\; E \qquad z = N \;\equiv\; F}{x = MN \;\equiv\; \exists y \exists z (E \wedge F \wedge y\,z\,x)}$$

falls xyz linear und $\mathcal{V}(xyz)$ disjunkt zu $\mathcal{FV}(MN)$

Abbildung 4.2: Einbettung des strikten λ-Kalküls

Im allgemeinen kann es zu festem x und M mehrere E geben, für die $x = M \equiv E$ gilt. Dies liegt an der Einführung von neuen lokaler Variablen und ist der Grund dafür, daß wir benannte λ-Ausdrücke nicht direkt als Ausdrücke von Δ definieren.

Proposition 4.1.2 *Falls $M \equiv N$, $u = M \equiv E$ und $u = N \equiv F$, dann folgt $E \equiv F$.*

Beweis. Einfach mit Induktion über Ableitungen von $M \equiv N$. Der Fall $M \equiv M$ kann mit Induktion über die Struktur von M gezeigt werden. □

Somit können wir benannte λ-Ausdrücke $u = M$ bis auf strukturelle Kongruenz als Elemente von Δ auffassen.

Proposition 4.1.3 *Alle Ausdrücke der Form $u = M$ sind zulässig.*

Den Beweis führen wir in Abschnitt 4.3.

Satz 4.1.4 (Adäquatheit) *Sei M geschlossen. Dann stimmt die Anzahl der (β_e)-Schritte in strikten Berechnungen von M mit der Anzahl von (A)-Schritten in Berechnungen von $u = M$ überein.*

Für den Beweis verweisen wir auf Abschnitt 4.4. Für die Längenmaße der beiden Kalküle bedeutet dies, daß $l_{\lambda_e}(M) = l^A_\delta(M)$ für alle $M \in \Lambda^{cl}$ gilt. Dies wiederum ist äquivalent dazu, daß für alle u die Einbettung $\Phi_u : (\Lambda^{cl}, \equiv, \to_e) \to \delta$ mit $\Phi_u(M) = (u = M)$ adäquat bezüglich der Längenmaße l_{λ_e} und l^A_δ ist.

Korollar 4.1.5 *Sei M geschlossen. Dann terminiert M genau dann im strikten λ-Kalkül, wenn $u = M$ im δ-Kalkül terminiert.*

Beweis. Wegen der Uniformität von l_δ (Satz 3.1.6) terminiert $u = M$ im δ-Kalkül genau dann, wenn $l_\delta(u = M) < \infty$. Da $\to_E$ terminiert und wegen der Uniformität von l_δ und l^A_δ (Satz 3.1.6) ist $l_\delta(u = M) < \infty$ äquivalent zu $l^A_\delta(u = M) < \infty$. Wegen Satz 4.1.4 ist dies wiederum gleichbedeutend mit $l_{\lambda_e}(M) < \infty$. Und da auch der strikte λ-Kalkül uniform (Satz 4.1.1) ist, ist letzteres äquivalent zur Terminierung von M. □

Wir illustrieren die Adäquatheitsaussage, indem wir eine Berechnung von $u = Copy\ (I\,I)$ im δ-Kalkül mit A-Länge drei vorstellen.

$$\begin{array}{rl}
u = Copy\ (I\,I) & \equiv \quad \exists y_1 \exists z_1 (y_1 = Copy \wedge z_1 = I\,I \wedge y_1\, z_1\, u) \\
(\to_A \circ \to_G) & \exists z_1 (u = z_1 z_1 \wedge z_1 = I\,I) \\
& \equiv \quad \exists z_1 (\exists y_3 \exists z_3 (y_3 = z_1 \wedge z_3 = z_1 \wedge y_3\, z_3\, u) \wedge z_1 = I\,I) \\
& \to_E \quad \exists z_1 (\exists z_3 (z_3 = z_1 \wedge z_1\, z_3\, u) \wedge z_1 = I\,I) \\
& \to_E \quad \exists z_1 (z_1\, z_1\, u \wedge z_1 = I\,I) \\
& \equiv \quad \exists z_1 (z_1\, z_1\, u \wedge \exists y_4 \exists z_4 (y_4 = I \wedge z_4 = I \wedge y_4\, z_4\, z_1)) \\
(\to_A \circ \to_G) & \exists z_1 (z_1\, z_1\, u \wedge \exists z_4 (z_4 = I \wedge z_1 = z_4)) \\
& \to_E \quad \exists z_4 (z_4\, z_4\, u \wedge z_4 = I) \\
& \to_A \quad \exists z_4 (u = z_4 \wedge z_4 = I)
\end{array}$$

Wir machen darauf aufmerksam, daß dieses Beispiel genau den gleichen Verdopplunkseffekt aufzeigt wie $Square\,(2 * 3)$ in der Einleitung. Ferner können wir auch am Beispiel von $Copy\ (I\,I)$ sehen, daß Reduktion im δ-Kalkül flexibler ist, als im strikten λ-Kalkül. Denn in der zweiten Zeile der voranstehenden Berechnung gibt es zwei reduzierbare Applikationen. Im Gegensatz dazu ist die Reduktion von $Copy\ (I\,I)$ im strikten λ-Kalkül deterministisch festgelegt, weil dieser funktionale Argumente stets vor Applikation auswertet.

4.2 Nicht-strikte funktionale Berechnung

Nicht-strikte funktionale Berechnung wird zum Beispiel von den Programmiersprachen Haskell [HWA+90], Miranda [Tur85], Id [Nik91] und Concurrent Clean [EHN+93] realisiert. Der nicht-strikte λ-Kalkül ist ein Terminierungsmodell von nicht-strikter funktionaler Berechnung, aber kein Komplexitätsmodell.

Im folgenden wiederholen wir zuerst die Definition des nicht-strikten λ-Kalküls ("lazy λ-calculus") und betten ihn anschließend in den strikten λ-Kalkül und in den δ-Kalkül ein. Die erste Einbettung erhält Terminierung und Komplexität und die zweite nur Terminierung. Dies liegt daran, daß die Einbettung in den δ-Kalkül "Sharing" einführt, so daß die Berechnung von mehrfach benötigten funktionalen Argumenten geteilt wird. Dies macht den δ-Kalkül zu einem Komplexitätsmodell für nicht-strikte funktionale Berechnung.

Die Ausdrücke und die Kongruenz des nicht-strikten und des strikten λ-Kalküls stimmen überein. Nicht-strikte Reduktion $\rightarrow_l$ definieren wir in Abbildung 4.3. Als nicht-strikten λ-Kalkül bezeichnen wir das Tripel $\lambda_l = (\Lambda, \equiv, \rightarrow_l)$.

$$(\beta) \quad (\lambda x.M)N \;\rightarrow_l\; M[N/x] \quad \text{falls } \mathcal{FV}(N) \cap \mathcal{BV}(M) = \emptyset$$

$$(Func) \quad \frac{M \rightarrow_l M'}{MN \rightarrow_l M'N}$$

$$(Congr) \quad \frac{M_1 \equiv M_2 \qquad M_2 \rightarrow_l N_2 \qquad N_2 \equiv N_1}{M_1 \rightarrow_l N_1}$$

Abbildung 4.3: Reduktion des nicht-strikten λ-Kalküls

Durch Anwendung von $\rightarrow_l$ werden funktionale Argumente, die nicht benötigt werden, weggeworfen. Dies geschieht zum Beispiel in $(\lambda x.y)\, Bomb \rightarrow_l y$, falls x und y verschieden sind. Hier finden wir das Beispiel einer konstanten Funktion $Const$ aus der Einleitung wieder. Das Wegwerfen von Argumenten ist dafür verantwortlich, daß sich das Terminierungsverhalten des strikten und nicht-strikten λ-Kalküls unterscheiden.

Mehrfach benötigte funktionale Argumente werden im nicht-strikten λ-Kalkül mehrfach ausgewertet, wie unser Standardbeispiel zeigt:

$$Copy\,(II) \;\rightarrow_l\; (II)\,(II) \;\rightarrow_l\; I\,(II) \;\rightarrow_l\; II \;\rightarrow_l\; I\,.$$

Aus diesem Grund ist der nicht-strikte λ-Kalkül nicht zur Modellierung der Komplexität von nicht-strikter Berechnung geeignet. Dennoch hat der nicht-strikte λ-Kalkül die typischen Eigenschaften eines Berechnungskalküls.

Satz 4.2.1 *Der nicht-strikte λ-Kalkül ist uniform konfluent.*

Beweis. Jeder Ausdruck M erlaubt im nicht-strikten λ-Kalkül höchstens eine Anwendung von (β). Reduziert M nach N_1 und N_2, so müssen N_1 und N_2 identisch bis auf α-Konversion sein. □

4.2.1 Der nicht-strikte im strikten λ-Kalkül

Wir zeigen nun, wie sich das Terminierungsverhalten von nicht-strikter Berechnung im strikten λ-Kalkül darstellen läßt. Bevor wir den allgemeinen Fall angehen, illustrieren wir dies am Beispiel eines funktionalen Konditionals. Dieses fassen wir

als dreistellige Funktion auf, die als Argumente eine boolsche Bedingung B, einen *then*-Zweig M und einen *else*-Zweig N entgegennimmt.

$$\mathsf{if}\ \mathit{Bool}\ \mathsf{then}\ M\ \mathsf{else}\ N\ \mathsf{fi}\ ,$$

Die Bedingung wollen wir strikt berechnen und die Zweige nicht-strikt, also nur bei nur Bedarf. Die Idee für die Kodierung der Konditionals ist die Berechnung von M und N zu verzögern, indem wir über diese abstrahieren und dann bei Bedarf applizieren.

Sei V eine beliebige Abstraktion, die wir zum Anstoßen von verzögerten Berechnung verwenden werden und $\_$ stets eine frische Variable. Dann definieren wir nicht-strikte Wahrheitswerte $True$ und $False$ durch:

$$True \;=\; \lambda x.\lambda y.\, x\mathsf{V} \qquad \text{und} \qquad False \;=\; \lambda x.\lambda y.\, y\mathsf{V}\,.$$

Diese selektieren also nicht nur eines ihrer Argumente, sondern stoßen auch dessen Berechnung an. Dann leistet die folgende Kodierung das Gewünschte:

$$\mathsf{if}\ \mathit{Bool}\ \mathsf{then}\ M\ \mathsf{else}\ N\ \mathsf{fi} \;=\; (\mathit{Bool}\ (\lambda\_.M))\ (\lambda\_.N)$$

Nun gibt es genau die folgenden Ableitungen, in denen jeweils nur der benötigte Zweig des Konditionals berechnet wird:

$$\begin{array}{rll}
\mathsf{if}\ True\ \mathsf{then}\ M\ \mathsf{else}\ N\ \mathsf{fi} & \to_e & (\lambda y.(\lambda_.M)\,\mathsf{V})\ (\lambda_.N) \\
& \to_e & (\lambda_.M)\,\mathsf{V} \\
& \to_e & M \\
\mathsf{if}\ False\ \mathsf{then}\ M\ \mathsf{else}\ N\ \mathsf{fi} & \to_e^* & (\lambda y.y\,\mathsf{V})\ (\lambda_.N) \\
& \to_e & (\lambda_.N)\,\mathsf{V} \\
& \to_e & N
\end{array}$$

Diese Idee wird zum Beispiel zur Implementierung von *delay* und *force* in Scheme [CR91] verwendet. Auch die Einbettung des nicht-strikten in den strikten λ-Kalkül in Abbildung 4.4 basiert genau auf der gleichen Idee. Dabei steht wiederum jedes Auftreten von $\_$ für eine frische Variable und V für eine beliebige Abstraktion.

$$[\![x]\!] \;\equiv\; x \qquad \frac{[\![M]\!] \equiv M'}{[\![\lambda x.M]\!] \;\equiv\; \lambda\_.\lambda x.M'\mathsf{V}} \qquad \frac{[\![M]\!] \equiv M' \qquad [\![N]\!]N'}{[\![M\,N]\!] \;\equiv\; \lambda\_.(M'\mathsf{V})\,N'}$$

Abbildung 4.4: Einbettung von λ_l in λ_e

Alle Ausdrücke der Form $[\![M]\!]$ sind irreduzibel, enthalten aber eine verzögerte Berechnung von M, die durch Applikation angestoßen werden kann.

Wir formulieren nun die Adäquatheit dieser Einbettung bezüglich Terminierung. Wie bereits vorher behauptet gilt sogar Adäquatheit bezüglich Komplexität. Dazu müßten wir aber ein geeignetes r-Längenmaß definieren, worauf wir an dieser Stelle verzichten wollen.

Satz 4.2.2 (Adäquatheit) *Sei* V *eine beliebige Abstraktion und* M *geschlossen. Dann terminiert die nicht-strikte Berechnung von* M *genau dann, wenn die strikte Berechnung von* $[\![M]\!]\mathsf{V}$ *terminiert. Dabei gilt* $l_{\lambda_l}(M) \leq l_{\lambda_e}([\![M]\!]\mathsf{V})$.

Der Beweis des Satzes läßt mit Hilfe von Satz 2.7.5 führen. Die Idee dabei ist, solche β_e-Schritte auszublenden, die lediglich für die nicht-strikte Kontrolle verantwortlich sind, um Adäquatheit bezüglich Komplexität nachzuweisen. Obwohl der Beweis nur etwa zwei bis drei Seiten umfaßt, führen wir ihn hier nicht aus.

4.2.2 Der nicht-strikte λ-Kalkül im δ-Kalkül

Im δ-Kalkül läßt sich bedarfsgesteuerte Berechnung mit dem richtigen Komplexitäts- und Terminierungsverhalten darstellen. Wir illustrieren dies zuerst anhand eines funktionalen Konditionals bevor wir den allgemeinen Fall behandeln. Dieses stellen wir als vierstellige relationale Prozedur dar, welche sich wie eine dreistellige funktionale Prozedur verhält, die strikt im ersten und nicht-strikt in den beiden anderen Argumenten ist.

$$out = (\mathsf{if}\ bool\ \mathsf{then}\ II\ \mathsf{else}\ (II)\ I\ \mathsf{fi})\ .$$

Sobald *bool* in einem imaginären Kontext an $True$ gebunden wird, soll der *then*-Zweig II berechnet werden. Eine Bindung von *bool* an $False$ soll zur Auswertung des *else*-Zweiges $(II)\ I$ führen.

Auch für dieses Beispiel sind Kontexte hilfreich. Unter einem Kontext T verstehen wir wiederum einen δ-Ausdruck mit einem Loch $\bullet$. Der Ausdruck $T[\![E]\!]$ entsteht durch textuelles Ersetzen von $\bullet$ in T durch E. Es gilt $E \to F$ im Kontext T genau dann, wenn $T[\![E]\!] \to T[\![F]\!]$. Für eine formal vollständige Definition von Kontexten verweisen wir auf Abschnitt 5.2.2.

Zur Darstellung des Konditionals im δ-Kalkül verfeinern wir die Idee, die wir bereits zur Darstellung im strikten λ-Kalkül verwendet haben. Wiederum stellen wir die Zweige des Konditionals als verzögerte Ausdrücke dar, deren Berechnung bei Bedarf angestoßen werden kann. Dazu assoziieren wir mit jedem der Zweige eine eigene logische Variable. Die Zweige selbst stellen wir so dar, daß sie auf die Bindung der assoziierten logischen Variable an eine strikte λ-Identität I warten. Im Gegensatz zur Darstellung im strikten λ-Kalkül suspendieren nicht benötigte Zweige auf ewig und werden nicht explizit weggeworfen.

Zur Kodierung des *then*-Zweiges II definieren wir einen Ausdruck $y \cdot s = II$, der die Berechnung von $y = II$ anstößt, sobald s im Kontext an die λ-Identität I gebunden wird:

$$y \cdot s = II \;\equiv\; \exists u \exists u' \exists v\,(u = I \wedge v = I \wedge s\,u\,u' \wedge u'\,v\,y)$$

Wir fordern dabei, daß alle auftretenden Variablen paarweise verschieden sind. Dies stellt sicher, daß $y \cdot s = II$ suspendiert und daß durch Reduktion keine Inkonsistenzen auftreten. In einem Kontext der Form $\exists s\,(s = I \wedge \bullet \wedge E)$ mit einem beliebigen E gilt dann:

$$\begin{array}{rll} y \cdot s = II & \to_A & \exists u \exists u' \exists v\,(u = I \wedge v = I \wedge u' = u \wedge u'\,v\,y) \\ & \to_E & \exists u \exists v\,(u = I \wedge v = I \wedge u\,v\,y) \\ & \to_A & \exists u \exists v\,(u = I \wedge v = I \wedge y = v) \\ & \to_G & \exists v\,(y = v \wedge v = I) \end{array}$$

Mit der gleichen Technik können wir auch $z \cdot t = (II)\,I$ als verzögerten Ausdruck definieren, der eine Berechnung von $y = (II)\,I$ anstößt, sobald t an die λ-Identität I gebunden wird.

Weiterhin benötigen wir boolsche Werte $True$ und $False$, die nicht nur einen Zweig selektieren, sondern auch dessen Berechnung anstoßen. Zu diesem Zweck erhalten $True$ auch $False$ als Eingabeargumente nicht nur Referenzen auf die Zweige, sondern auch Referenzen auf die assoziierten logischen Variablen. Bei Bedarf eines Zweiges binden sie die zugehörige logische Variable an die λ-Identität. In Kontexten der Form $\exists r\,(r = I \wedge \bullet \wedge E)$ leisten die folgenden Definitionen von $True$ und $False$ das Gewünschte:

$$\begin{array}{l} y\,s\,z\,t\,out\,/\,(out = y \wedge s = r) \\ y\,s\,z\,t\,out\,/\,(out = z \wedge t = r) \end{array}$$

Die gesuchte Darstellung des Konditionals hat nun die folgende Form:

$$\begin{array}{rl} out = & (\mathsf{if}\ bool\ \mathsf{then}\ II\ \mathsf{else}\ (II)\,I\ \mathsf{fi} \\ \equiv & \exists s \exists t \exists y \exists z\,(bool\,y\,s\,z\,t\,out \wedge y \cdot s = II \wedge z \cdot t = (II)\,I) \end{array}$$

In Kontexten T_{true} und T_{false} der folgenden Bauart

$$\begin{array}{rcl} T_{true} & = & \exists bool \exists r\,(r = I \wedge bool\colon True \wedge \bullet \wedge E) \\ T_{false} & = & \exists bool \exists r\,(r = I \wedge bool\colon False \wedge \bullet \wedge E) \end{array}$$

reduziert das Konditional wie gewünscht. Zum Beispiel erhalten wir im Kontext von T_{true} die folgende Berechnung, wobei wir $Dead$ abkürzend für $\exists z \exists t(z \cdot t = (II)\,I)$ schreiben.

$$\begin{array}{rl} out = & (\mathsf{if}\ bool\ \mathsf{then}\ II\ \mathsf{else}\ (II)\,I\ \mathsf{fi}) \\ \to_A & \exists s \exists t \exists y \exists z\,(out = y \wedge s = r \wedge y \cdot s = II \wedge z \cdot t = (II)\,I) \\ \to_E & \exists y(out = y \wedge y \cdot r = II) \wedge Dead \\ \to^* & \exists y(out = y \wedge \exists v\,(y = v \wedge v = I)) \wedge Dead \\ \to_E & \exists v(out = v \wedge v = I) \wedge Dead \end{array}$$

Wir kommen nun zur Einbettung des nicht-strikten λ-Kalküls im δ-Kalkül. Dazu kodieren nicht-strikte Prozeduren analog zu $True$ und $False$ im voranstehenden Beispiel stets so, daß sie nicht nur ihr Resultat berechnen sondern auch die Berechnung der Argumente anstoßen, die sie benötigen. Wird ein Argument mehrfach angefordert, dann wird die mit ihm assoziierte logische Variable mehrfach gebunden. Wir stellen Konsistenz dadurch sicher, daß wir solche logischen Variablen stets an dasselbe Auftreten einer λ-Identität binden.

Zur Definition der Einbettung von λ_l in δ benötigen wir erweiterte λ-Ausdrücke, die wir wiederum mit M und N bezeichnen.

$$M, N \ ::= \ x \cdot r \mid x \mid \lambda x.M \mid MN$$

Unter einem benannten verzögerten λ-Ausdruck $z \cdot r = M$ verstehen wir formal ein Tripel bestehend aus zwei Variablen z und r und einem erweiterten λ-Ausdruck M. In Abbildung 4.5 definieren wir eine Relation $\equiv$ zwischen solchen Tripeln und Ausdrücken von Δ.

$$x \cdot r = y \cdot s \ \equiv \ x = y \wedge s = r \qquad x \cdot r = y \ \equiv \ x = y$$

$$\frac{z \cdot r = M[y \cdot s/y] \ \equiv \ E}{x \cdot r = \lambda y.M \ \equiv \ x{:}y\,s\,z/E} \quad \text{falls } \{y, s\} \cap \mathcal{BV}(M) = \emptyset$$

$$\frac{y \cdot r = M \ \equiv \ E \qquad z \cdot s = N \ \equiv \ F}{x \cdot r = MN \ \equiv \ \exists y \exists y' \exists s \exists z (E \wedge F \wedge r\,y\,y' \wedge y'\,z\,s\,x)}$$

falls $\overline{w} = xyy'zrs$ linear und $\mathcal{V}(\overline{w})$ disjunkt zu $\mathcal{FV}(MN)$.

Abbildung 4.5: Einbettung von λ_l nach δ

Wir machen darauf aufmerksam, daß wir zur Darstellung des Konditionals nicht alle auftretenden Funktionen nicht-strikt übersetzt haben. Aus diesem Grund stimmen die dort verwendeten Ausdrücke $y \cdot s = II$ und $z \cdot t = (II)\,I$ nicht vollständig mit Ausdücken des Übersetzungsschemas überein.

Proposition 4.2.3 *Falls $M \equiv N$, $x \cdot r = M \equiv E$ und $x \cdot r = N \equiv F$, dann folgt $E \equiv F$.*

Beweis. Wie im strikten Fall mit Induktion über Ableitungen von $M \equiv N$. Der Fall $M \equiv M$ kann mit Induktion über die Struktur von M gezeigt werden. □

Proposition 4.2.4 *Für alle M ist $r = I \wedge x \cdot r = M$ zulässig.*

Den Beweis führen wir in Abschnitt 4.3. Der kritische Punkt ist dabei, daß assoziierte logische Variablen mehrfach gebunden werden können. Dies können wir an folgendem Beispiel beobachten, in dem $Copy$ wiederum für $\lambda y.yy$ steht. Bei Applikation von x im Kontext von

$$\exists r\,(r = I \wedge x \cdot r = Copy \wedge \bullet)$$

wird y zweimal angefordert und somit die zu y assoziierte logische Variable zweimal gebunden.

Eingebettete Ausdrücke enthalten sowohl zweistellige als auch dreistellige Abstraktionen und Applikationen. Die zweistelligen dienen dabei ausschließlich zur Kontrolle, also zur Applikation der strikten λ-Identität. Im folgenden bezeichnen wir Applikationen von dreistelligen Abstraktionen mit $\rightarrow_{A_3}$ und Applikationen von zweistelligen Abstraktionen mit $\rightarrow_{A_2}$. Mit $l_\delta^{A_3}$ bezeichnen wir das A_3-Längenmaß, das nur Applikationen von dreistelligen Abstraktionen zählt.

Vermutung 4.2.5 (Adäquatheit) *Sei M geschlossen. Dann terminiert M im nicht-strikten λ-Kalkül genau dann, wenn $\exists r\,(r = I \wedge x \cdot r = M)$ im δ-Kalkül terminiert. Das A_3-Längenmaß ist uniform und erfüllt $l_\delta^{A_3}(\exists r\,(r = I \wedge x \cdot r = M)) \leq l_{\lambda_l}(M)$.*

Somit ist Komplexität von nicht-strikter Berechnung im δ-Kalkül auf keinen Fall schlechter als im nicht-strikten λ-Kalkül. Ein analoges Resultat wird für den sogenannten "Call-By-Need" λ-Kalkül in [AFM+95] gezeigt. Dieser Kalkül verwendet *let*-Ausdrücke zur Darstellung von Referenzen und beschreibt nicht-strikte Kontrolle durch Auszeichnung spezieller Kontexte ("Evalulation Contexts"). Vermutlich gilt selbiges Resultat auch für den Kalkül in [Lau93].

Wir rechnen nun ein typisches Beispiel durch, nämlich die Applikation von $Copy\ (II)$, in dem durch Einbettung Komplexität echt verbessert wird. Allerdings sind die Kosten für Kontrollschritte relativ hoch. Dieses Beispiel zeigt ein inherentes Problem von nicht-strikter Berechnung, nämlich daß man für nicht-strikte Kontrolle im Vergleich zu strikter Kontrolle einen Effizienzverlust in Kauf nehmen muß. Häufig wird versucht dieses Problem durch Striktheitsanlyse zu umgehen [CC77].

$$\begin{aligned}
&\exists r\,(r = I \wedge out \cdot r = Copy\ (II)) \\
&\quad \rightarrow_{A_2} \quad \exists r\,(r = I \wedge \exists s_1 \exists y_1 \exists z_1\,(y_1 \cdot r = Copy \wedge z_1 \cdot s_1 = II \wedge y_1\, z_1\, s_1\, out)) \\
&\quad \rightarrow_{A_3} \quad \exists s_1 \exists y_1 \exists z_1 \exists r\,(r = I \wedge y_1 \cdot r = Copy \wedge z_1 \cdot s_1 = II \wedge \\
&\qquad\qquad\qquad\qquad out \cdot r = (z_1 \cdot s_1)\,(z_1 \cdot s_1)) \\
&\quad \rightarrow_{G} \quad \exists s_1 \exists z_1 \exists r\,(r = I \wedge z_1 \cdot s_1 = II \wedge out \cdot r = (z_1 \cdot s_1)\,(z_1 \cdot s_1))
\end{aligned}$$

Als Zwischenberechnung reduzieren wir nun $out \cdot r = (z_1 \cdot s_1)\,(z_1 \cdot s_1)$ im Kontext

von $\exists s_1 \exists z_1 \exists r\,(r = I \wedge z_1 \cdot s_1 = II \wedge \bullet)$:

$$\begin{array}{rl} out \cdot r = (z_1 \cdot s_1)\,(z_1 \cdot s_1) & \\ \rightarrow_{A_2} & \exists s_2 \exists y_2 \exists z_2 (y_2 \cdot r = z_1 \cdot s_1 \wedge z_2 \cdot s_2 = z_1 \cdot s_1 \wedge y_2\, z_2\, s_2\, out) \\ \equiv & \exists s_2 \exists y_2 \exists z_2 (y_2 = z_1 \wedge r = s_1 \wedge z_2 = z_1 \wedge s_2 = s_1 \wedge y_2\, z_2\, s_2\, out) \\ \rightarrow_E^* & r = s_1 \wedge z_1\, z_1\, s_1\, out \wedge s_1 = s_1 \end{array}$$

Nun wollen s_1 im Gesamtausdruck einschließlich Kontext eliminieren und $z_1 \cdot s_1 = II$ berechnen. Dazu reduzieren wir $\exists s_1 (r = s_1 \wedge z_1 \cdot s_1 = II)$ im Kontext von $\exists z_1 \exists r\,(r = I \wedge \bullet \wedge z_1\, z_1\, s_1\, out \wedge s_1 = s_1)$:

$$\begin{array}{rl} \exists s_1 (r = s_1 \wedge z_1 \cdot s_1 = II) & \\ \rightarrow_E & z_1 \cdot r = II \\ \rightarrow_{A_2} & \exists s_2 \exists y_2 \exists z_2 (y_2 \cdot r = I \wedge z_2 \cdot s_2 = I \wedge y_2\, z_2\, s_2\, z_1) \\ \rightarrow_{A_3} & \exists s_2 \exists y_2 \exists z_2 (y_2 \cdot r = I \wedge z_2 \cdot s_2 = I \wedge z_1 \cdot r = z_2 \cdot s_2) \\ \rightarrow_G & \exists s_2 \exists z_2 (z_2 \cdot s_2 = I \wedge z_1 = z_2 \wedge r = s_2) \\ \rightarrow_E^* & z_1 \cdot r = I \end{array}$$

Nun führen wir die Berechnung im vollständigen Kontext fort und applizieren z_1.

$$\begin{array}{rl} \exists s_1 \exists z_1 \exists r\,(r = I \wedge z_1 \cdot r = I \wedge z_1\, z_1\, s_1 out \wedge s_1 = s_1) & \\ \rightarrow_{A_3} & \exists s_1 \exists z_1 \exists r\,(r = I \wedge z_1 \cdot r = I \wedge out \cdot r = z_1 \cdot s_1 \wedge s_1 = s_1) \\ \equiv & \exists s_1 \exists z_1 \exists r\,(r = I \wedge z_1 \cdot r = I \wedge out = z_1 \wedge r = s_1 \wedge s_1 = s_1) \\ \rightarrow_E^* & \exists z_1 \exists r\,(r = I \wedge z_1 \cdot r = I \wedge out = z_1 \wedge r = r) \end{array}$$

In der Tat ist *out* nun an die nicht-strikte λ-Identität gebunden. Dazu haben wir insgesamt drei (A_3)-Schritte benötigt, genauso viele, wie (β)-Schritte in strikten Berechnungen von $Copy\ (II)$.

4.3 Inkonsistenzen

Wir zeigen nun, daß δ-Ausdrücke der Form $x = M$ und $x \cdot r = M$ stets zulässig sind. Dadurch beweisen wir die Propositionen 4.1.3 und 4.2.4.

Wir stellen zwei Zulässigkeitstests vor, die wir auch in einem einzigen kombinieren könnten. Der erste Test ist hinreichend für Ausdrücke $x = M$ aus der Einbettung des strikten λ-Kalküls und der zweite für Ausdrücke $x \cdot r = M$ aus der Einbettung des nicht-strikten λ-Kalküls.

Der erste Test stellt sicher, daß alle auftretenden Variablen höchstens einmal gebunden werden können. Der zweite Test unterscheidet statisch zwei Klassen von Variablen: Variablen der ersten Klasse dürfen höchstens einmal gebunden werden. Variablen der zweiten Klasse dürfen mehrfach gebunden werden, aber nur an ein

festes Auftreten einer λ-Identität. Variablen der ersten Klasse haben wir in dieser Arbeit durchgängig mit x, y, z, u, v, w bezeichnet und Variablen der zweiten Klasse mit r, s, t. Die beiden Tests könnten kombiniert werden, indem man die Klasse einer Variablen rät, anstatt Klassen durch durch Argumentpositionen festzulegen.

Der strikte λ-Kalkül

Bei der Übersetzung des strikten λ-Kalküls werden ausschließlich zweistellige Abstraktionen verwendet. Das erste Argument dient zur Eingabe und das zweite zur Ausgabe. Desweiteren werden Gleichungen gerichtet eingesetzt, so daß nur die Variablen, die links in Gleichungen stehen, gebunden werden können.

Diese Beobachtungen führt zu folgendem Zulässigkeitsbegiff: Ein Ausdruck E heißt e-zulässig, wenn eine Menge P von Variablen existiert, so daß ein Urteil der Form $P \ \triangleright\ E$ mit den Regeln in Abbildung 4.6 herleitbar ist.

$$\{z\} \ \triangleright\ x\,y\,z \qquad \{x\} \ \triangleright\ x = y \qquad \emptyset \ \triangleright\ \top$$

$$\frac{P \ \triangleright\ E}{\{x\} \ \triangleright\ x{:}y\,z/E} \quad \text{falls } P \subseteq \{z\} \qquad \frac{P \ \triangleright\ E}{P \setminus \{x\} \ \triangleright\ \exists x\,E}$$

$$\frac{P_1 \ \triangleright\ E_1 \qquad P_2 \ \triangleright\ E_2}{P_1 \cup P_2 \ \triangleright\ E_1 \wedge E_2} \quad \text{falls } P_1 \cap P_2 = \emptyset$$

Abbildung 4.6: e-Zulässigkeit

Es ist einfach nachzurechnen, daß für alle Ausdrücke E höchstens eine Menge P existiert mit $P \ \triangleright\ E$. Falls ein solches P existiert, nennen wir P die Menge der in E produzierten Variablen und bezeichnen sie mit $\mathcal{P}(E)$.

Lemma 4.3.1 *Die in E produzierten Variablen sind frei in E, d.h. $\mathcal{P}(E) \subseteq \mathcal{FV}(E)$.*

Beweis. Mit Induktion über Herleitungen von $P \ \triangleright\ E$. □

Proposition 4.3.2 *Alle Ausdrücke $x = M$ sind e-zulässig mit $\mathcal{P}(x = M) = \{x\}$.*

Beweis. Mit Induktion über Herleitungen von $x = M \equiv E$. □

Lemma 4.3.3 *Ist E e-zulässig, dann sind $\exists x\,E$ und alle Teilausdrücke von E e-zulässig.*

Beweis. Mit struktureller Induktion über E. □

Lemma 4.3.4 *e-Zulässigkeit ist invariant unter struktureller Kongruenz und Reduktion. Falls E zulässig und $E \to^* F$, dann gilt $\mathcal{P}(F) \subseteq \mathcal{P}(E)$.*

Den Beweis dieses Lemmas werden wir in Kürze führen.

Lemma 4.3.5 *e-zulässige Ausdrücke sind konsistent.*

Beweis. Wegen Lemma 4.3.3 reicht es zu zeigen, daß Ausdrücke der Form $x\colon A \wedge x\colon A'$ nicht e-zulässig sind. Dazu nehmen wir an, daß eine Herleitung eines Urteils $P \rhd x\colon A \wedge x\colon A'$ existiert. Diese muß von der Regel für Kompositon abstammen, welche die Herleitbarkeit von Urteilen der Form $\{x\} \rhd x\colon A$ und $\{x\} \rhd x\colon A'$ voraussetzt. Dies steht jedoch im Widerspruch zur Nebenbedingung der Kompositionsregel steht. □

Proposition 4.3.6 *e-zulässige Ausdrücke sind zulässig.*

Beweis. Sei nun E ein e-zulässiger Ausdruck und $\overline{x}$ eine Folge von Variablen. Wir müssen die Konsistenz aller F zeigen für die $\exists \overline{x} E \to^* F$ gilt. Wegen Lemma 4.3.3 ist $\exists \overline{x} E$ e-zulässig und somit wegen Lemma 4.3.4 auch F. Wegen Lemma 4.3.5 ist F auch konsistent. □

Beweis von Satz 4.1.3. Ausdrücke der Form $x = M$ sind e-zulässig nach Proposition 4.3.2, so daß aus Proposition 4.3.6 Zulässigkeit folgt. □

Zur Vervollständigung des Beweises zeigen wir nun Lemma 4.3.4, das sich auf folgende Aussage zurückführen läßt.

Lemma 4.3.7 *Sei E e-zulässig und $y \notin \mathcal{BV}(E)$. Falls $y \notin \mathcal{P}(E)$ oder $x \notin \mathcal{P}(E)$, dann ist $E[y/x]$ e-zulässig mit*

$$\mathcal{P}(E[y/x]) \;=\; \mathcal{P}(E)[y/x]\,.$$

Beweis. Mit Induktion über die Struktur von Ausdrücken E. Wir betrachten nur den schwierigsten Fall, die Regel für eine Komposition $E = E_1 \wedge E_2$. Dazu bemerken wir zuerst, daß sich e-Zulässigkeit und die Freiheitsbedingung auf die Teilausdrücke E_1 und E_2 vererben (Lemma 4.3.3). Wegen der Nebenbedingung der Regel für Komposition ist $\mathcal{P}(E_1) \cup \mathcal{P}(E_2)$ eine disjunkte Zerlegung von $\mathcal{P}(E)$, d.h.:

$$\mathcal{P}(E) \;=\; \mathcal{P}(E_1) \cup \mathcal{P}(E_2) \qquad \text{und} \qquad \mathcal{P}(E_1) \cap \mathcal{P}(E_2) \;=\; \emptyset \tag{4.1}$$

Die Induktionsvoraussetzung angewendet auf E_1 und E_2 stellt die e-Zulässigkeit von $E_1[y/x]$ und $E_2[y/x]$ sicher, also auch:

$$\mathcal{P}(E_1[y/x]) = \mathcal{P}(E_1)[y/x] \quad \text{und} \quad \mathcal{P}(E_2[y/x]) = \mathcal{P}(E_2)[y/x] \tag{4.2}$$

Daraus folgt die nachstehende Disjunktheitseigenschaft, wie wir gleich ausführen werden.

$$\mathcal{P}(E_1)[y/x] \cap \mathcal{P}(E_2)[y/x] \;=\; \emptyset \tag{4.3}$$

Dies ist im Fall $x \notin \mathcal{P}(E)$ trivial; anderenfalls gilt $x \in \mathcal{P}(E)$ und nach Voraussetzung des Lemmas $y \notin \mathcal{P}(E)$. Wegen der obigen Zerlegung (4.1) können wir ohne Beschränkung der Allgemeinheit $x \in \mathcal{P}(E_1)$ und $x \notin \mathcal{P}(E_2)$ annehmen. Dann folgt Formel (*4.3*) aus

$$\mathcal{P}(E_1)[y/x] = (\mathcal{P}(E_1) \setminus \{x\}) \cup \{y\} \quad \text{und} \quad \mathcal{P}(E_2)[y/x] = \mathcal{P}(E_2)\,.$$

Unter Verwendung von (*4.3*) und (*4.2*) erhalten wir die e-Zulässigkeit von $E[y/x]$ wie folgt:

$$\begin{array}{rcl} \mathcal{P}(E[y/x]) & = & \mathcal{P}(E_1[y/x] \wedge E_2[y/x]) \\ & = & \mathcal{P}(E_1[y/x]) \cup \mathcal{P}(E_2[y/x]) \\ & = & \mathcal{P}(E_1)[y/x] \cup \mathcal{P}(E_2)[y/x] \\ & = & (\mathcal{P}(E_1) \cup \mathcal{P}(E_2))[y/x] \\ & = & \mathcal{P}(E)[y/x] \end{array}$$

□

Beweis von Lemma 4.3.4. Wir führen hier lediglich die Aussage über Reduktion aus und überlassen diejenige für die strukturelle Kongruenz dem Leser. Den Nachweis führen wir mit Induktion über Herleitungen von $E \to F$. Wir führen jedoch auch hier nur die schwierigsten Fälle aus, die Axiome für Elimination (E) und Applikation (A).

Im Falle von (E) setzen wir die e-Zulässigkeit von $\exists x \exists y (x = y \wedge E)$ und $y \notin \mathcal{BV}(E)$ voraus und müssen die e-Zulässigkeit für $\exists y\, E[y/x]$ nachweisen. Nun gilt $x \notin \mathcal{P}(E)$, so daß wir Lemma 4.3.7 anwenden können, woraus die e-Zulässigkeit von $E[y/x]$ und $\mathcal{P}(E[y/x]) = \mathcal{P}(E)[y/x]$ folgt. Dann ist auch $\exists y\, E[y/x]$ e-zulässig und es gilt:

$$\mathcal{P}(\exists y E[y/x]) \;=\; \mathcal{P}(E)[y/x] \setminus \{y\} \;=\; \mathcal{P}(E) \setminus \{x, y\} \;=\; \mathcal{P}(\exists x \exists y (x = y \wedge E))\,.$$

Im Falle von (A) setzen wir die e-Zulässigkeit von $x{:}y\,z/E \wedge x\,u\,v$ und die Nebenbedingungen $\{u, v\} \cap \mathcal{BV}(E) = \emptyset$ und $y \neq z$ voraus. Dann zeigen wir, daß $x{:}y\,z/E \wedge E[uv/yz]$ e-zulässig ist und höchstens x und v produziert.

Wir können annehmen, daß u und z verschieden sind, da sich dies durch Umbenennug von z erreichen läßt. Diese Annahme stellt $E[uv/yz] = E[u/y][v/z]$ sicher, wodurch wir simultane Substitutionen eliminieren können.

Aus der e-Zulässigkeitsregel für Abstraktionen folgt $\mathcal{P}(E) \subseteq \{z\}$ und somit $y \notin \mathcal{P}(E)$. Nun können wir Lemma 4.3.7 anwenden und erhalten die e-Zulässigkeit von $E[u/y]$ mit $\mathcal{P}(E[u/y]) = \mathcal{P}(E)[u/y] = \mathcal{P}(E)$. Im Fall $v \neq z$ gilt $v \notin \mathcal{P}(E)$, so daß wir Lemma 4.3.7 erneut einsetzen können. Daraus folgt die e-Zulässigkeit von $E[u/y][v/z]$ mit $\mathcal{P}(E[u/y][v/z]) = \mathcal{P}(E)[v/z] \subseteq \{v\}$. Im Fall $v = z$ gilt $E[u/y][v/z] = E[u/y]$ und somit $\mathcal{P}(E[u/y][v/z]) \subseteq \{v\}$. Folglich ist $x{:}y\,z/E \wedge E[u/y][v/z]$ e-zulässig mit $\mathcal{P}(x{:}y\,z/E \wedge E[u/y][v/z]) \subseteq \{x, v\}$. □

Der nicht-strikte λ-Kalkül

Wir beschreiben nun einen Zulässigkeitstest, der für benannte verzögerte λ-Ausdrücke $x \cdot r = M$ hinreichend ist.

Seien tt und ff die boolschen Wahrheitswerte *true* und *false* mit Konjunktion $\wedge$ und Disjunktion $\vee$. Wir nennen einen Ausdruck E l-zulässig, falls zwei Variablenmengen P und T und ein boolscher Wert B existieren, so daß ein Urteil der Form $(P, T, B) \rhd E$ mit den Regeln in Abbildung 4.7 herleitbar ist. Die Menge P enthält die Variablen der ersten Klasse in E, also solche die höchstens einmal gebunden werden dürfen. Die Menge[1] T enthält die Variablen der zweiten Klasse in E. Diese können mehrfach gebunden werden, aber stets an dasselbe Auftreten einer λ-Identität. Ist ein Urteil $(P, T, B) \rhd E$ herleitbar, dann gilt $B = tt$ genau dann, wenn eine Variablen aus T in E an eine λ-Identität I gebunden ist. Bindungen von Variablen der zweiten Klasse in Rümpfen von Abstraktionen sind nicht gestattet.

Lemma 4.3.8 *Für alle herleitbaren Urteile $(P, T, B) \rhd E$ sind P und T disjunkt.*

Beweis. Mit Induktion über Herleitungen von Urteilen $(P, T, B) \rhd E$. □

Proposition 4.3.9 *Alle Ausdrücke der From $E \equiv r{=}I \wedge x{\cdot}r{=}M$ sind l-zulässig mit $(\{x\}, \{r\}, tt) \rhd E$.*

Beweis. Mit Induktion über die Definition von $x \cdot r = M$ können wir für alle x und M zeigen, daß Urteile der Form $(\{x\}, \{r\}, ff) \rhd x \cdot r = M$ herleitbar sind. □

Lemma 4.3.10 *Ist E l-zulässig, dann sind $\exists x\,E$ und alle Teilausdrücke von E l-zulässig.*

[1] T steht für "Trigger", denn über die Variablen in dieser Menge können wir verzögerte Berechnungen anstoßen.

$$(\{z\},\{r\},ff) \rhd x\,y\,r\,z \qquad (\emptyset,\{r\},tt) \rhd r{:}y\,z/z = y \qquad (\emptyset,\emptyset,ff) \rhd \top$$

$$(\{x\},\emptyset,ff) \rhd x = y \qquad (\emptyset,\{r,s\},ff) \rhd r = s$$

$$\frac{(P,T,ff) \rhd E}{(\{x\},T\setminus\{r\},ff) \rhd x{:}y\,r\,z/E} \text{ falls } P \subseteq \{z\}$$

$$\frac{(P,T,B) \rhd E}{(P\setminus\{x\},T,B) \rhd \exists x\,E} \text{ falls } x \notin T \qquad \frac{(P,T,B) \rhd E}{(P,T\setminus\{r\},B) \rhd \exists r\,E} \text{ falls } r \notin P$$

$$\frac{(P_1,T_1,B_1) \rhd E_1 \qquad (P_2,T_2,B_2) \rhd E_2}{(P_1\cup P_2, T_1\cup T_2, B_1\vee B_2) \rhd E_1\wedge E_2} \begin{array}{l}\text{falls } P_1\cap P_2=\emptyset,\ B_1\wedge B_2 = ff,\\ P_1\cap T_2=\emptyset \text{ und } P_2\cap T_1=\emptyset\end{array}$$

Abbildung 4.7: l-Zulässigkeit

Beweis. Mit Induktion über die Struktur von E. □

Lemma 4.3.11 *l-Zulässigkeit ist abgeschlossen unter Kongruenz und Reduktion.*

Beweis. Der Fall der Kongruenz ist einfach. Im Falle der Reduktion gibt es zwei Arten von Substitutionen: Entweder sind beide beteiligten Variablen in T oder nicht. Der erste Fall ist einfach, da die Kompositionsregel als Nebenbedingung nicht die Disjunktheit von T_1 und T_2 fordert. Der Nachweis des zweiten Falls verläuft analog zum Beweis für e-Zulässigkeit. □

Lemma 4.3.12 *l-zulässige Ausdrücke sind konsistent.*

Beweis. Wegen Lemma 4.3.10 reicht es zu zeigen, daß $x{:}\,A_1 \wedge x{:}\,A_2$ nicht l-zulässig ist. Dazu führen wir die Annahme zum Widerspruch, daß ein Urteil der Form $(P,T,B) \rhd x{:}\,A_1 \wedge x{:}\,A_2$ herleitbar ist. Die letzte verwendete Regel zur Herleitung muß die Regel für Komposition sein. Somit gibt es Herleitungen der Bauart $(P_1,T_1,B_1) \rhd x{:}\,A_1$ und $(P_2,T_2,B_2) \rhd x{:}\,A_2$.

Wir betrachten zuerst den Fall $x \in T$. Wegen $T = T_1 \cup T_2$ gilt dann $x \in T_1$ oder $x \in T_2$. Aus Symmetriegründen können wir ohne Beschränkung der Allgemeinheit $x \in T_1$ annehmen. Nach Lemma 4.3.8 sind T_1 und P_1 disjunkt und wegen der Nebenbedingung der Kompositionsregel auch T_1 und P_2. Daraus folgt $x \notin P_1$ und $x \notin P_2$. Somit müssen die Herleitungen für die Urteile $(P_1,T_1,B_1) \rhd x{:}\,A_1$ und $(P_2,T_2,B_2) \rhd x{:}\,A_2$ beide das Axiom für zweistellige Abstraktionen verwendet haben. Daraus folgt dann $B_1 = B_2 = tt$ im Widerspruch zu $B_1 \wedge B_2 = ff$.

Im zweiten Fall gilt $x \notin T$. Dann kann weder das Urteil $(P_1,T_1,B_1) \rhd x{:}\,A_1$ noch das Urteil $(P_2,T_2,B_2) \rhd x{:}\,A_2$ vom Axiom für zweistellige Abstraktionen abstammen. Sie müssen also beide aus der Regel für dreistellige Abstraktionen hergeleitet

worden sein. Daraus folgt $x \in P_1$ und $x \in P_2$ im Widerspruch zur Disjunktheit von P_1 und P_2, einer Nebenbedingung der Regel für dreistellige Abstraktionen. □

Proposition 4.3.13 *l-zulässige Ausdrücke sind zulässig.*

Beweis. Folgt wie zum Fall der e-Zulässigkeit unmittelbar, hier jedoch unter Verwendung der Lemmata 4.3.10, 4.3.11 und 4.3.12. □

Beweis von Proposition 4.2.4. Ausdrücke der Form $r = I \wedge x \cdot r = M$ sind l-zulässig nach Proposition 4.3.9, so daß aus Proposition 4.3.13 Zulässigkeit folgt. □

4.4 Adäquatheitsbeweise

Wir stellen im folgenden den Adäquatheitsbeweis für die Einbettung des strikten λ-Kalküls in den δ-Kalkül vor, wodurch wir Satz 4.1.4 beweisen. Den Nachweis der Korrektheit der Einbettung des nicht-strikten λ-Kalküls (Vermutung 4.2.5) bleiben wir schuldig.

Die Idee des Beweises ist, Milners Technik basierend auf explizite Substitutionen [Mil92] mit Uniformität zu kombinieren. Alle Ideen die wir im Bezug auf Uniformität benötigen, haben wir bereits in Kapitel 2 herausfaktorisiert. Insbesondere machen wir Gebrauch von Lägensimulationen indem wir Satz 2.7.4 verwenden.

Wir betrachten im folgenden die durch u parametrisierten Einbettungen Φ_u des strikten λ-Kalküls in den δ-Kalkül:

$$\Phi_u(M) \;\equiv\; u = M\ .$$

Wir motivieren zuerst an einem Beispiel, wie eine Längensimulation für Φ_u aussehen könnte. Dazu stellen wir die Symmetrie zwischen einer Berechnung von $Copy\ (II)$ im strikten λ-Kalkül und der analogen Berechnung im δ-Kalkül heraus. Im Falle des strikten λ-Kalküls machen wir Substitutionen explizit:

$$\begin{array}{lll} Copy\ (II) & \to_e & (Copy\ v_1)\,[I/u_1][I/v_1] \\ & \to_e & (v_1 v_1)\,[Copy/u_2][I/u_1][I/v_1] \\ & \to_e & v_1\,[Copy/u_2][I/u_1][I/v_1] \end{array}$$

Durch explizite Substitutionen können wir also Kopieren und Annullieren (G) von Abstraktionen beschreiben. Für die analoge Berechnung im δ-Kalkül übernehmen

Kontexte die Rolle von expliziten Substitutionen.

$$\begin{array}{rll} u = Copy\ (II) & \to_A & T_1[\![u = Copy\ v_1]\!] \\ & & T_1 = \exists u_1 \exists v_1 (\bullet \wedge u_1 = I \wedge v_1 = I) \\ & \to_A & T_1[T_2[\![\exists v_2 (u = v_2 v_2 \wedge v_2 = v_1)]\!]] \\ & & T_2 = \exists u_2 (\bullet \wedge u_2 = Copy) \\ & \to_E^* & T_1[T_2[\![v_1\, v_1\, u]\!]] \\ & \to_A & T_1[T_2[\![u = v_1]\!]] \end{array}$$

Eine genauso exakte Symmetrie läßt sich bei der Berechnung von $(FirstI)\,I$ beobachten, wobei $First$ wiederum für $\lambda x.\lambda y.\, x$ steht.

$$\begin{array}{rll} (First\ I)\ I & \to_e & ((\lambda y.v_1)\ I)\,[First/u_1][I/v_1] \\ & \to_e & v_1\,[\lambda y.v_1/u_2][I/v_2][First/u_1][I/v_1] \end{array}$$

Wiederum entsprechen Kontexte im Falle von δ expliziten Substitutionen für λ_e.

$$\begin{array}{rll} u = (First\ I)\ I & \to_A & T_1[\![u = (\lambda y.v_1)\ I]\!] \\ & & T_1 = \exists u_1 \exists v_1 (\bullet \wedge u_1 = First \wedge v_1 = I) \\ & \to_A & T_1[T_2[\![u = v_1]\!]] \\ & & T_2 = \exists u_2 \exists v_2 (\bullet \wedge u_2 = \lambda y.v_1 \wedge v_2 = I) \end{array}$$

Zur Formalisierung dieser Symmetrie benötigen wir eine passende Notation für explizite Substitutionen und müssen deren Zusammenhang zu Kontexten erfassen. Dazu definieren wir die Anwendung des sequentiellen Substitutionsoperator $\langle \overline{V}/\overline{v} \rangle$ auf M durch

$$M\langle \overline{V}/\overline{v} \rangle \;=\; M[V_1/v_1] \ldots [V_n/v_n]\,,$$

falls $\overline{v} = (v_1, \ldots, v_n)$ und $\overline{V} = (V_1, \ldots, V_n)$. Desweiteren benötigen wir spezielle Kontexte $T_{\overline{v}=\overline{V}}$, die wir wie folgt definieren:

$$T_{\overline{v}=\overline{V}} \;=\; \exists \overline{v} (\overline{v} = \overline{V} \wedge \bullet)\,.$$

Nun koppeln wir explizite Substitution mit den passenden Kontexten; wir nennen $(N, \overline{V}, \overline{v})$ eine u-Darstellung von (M, E), falls gilt:

1. $M \equiv N\langle \overline{V}/\overline{v} \rangle$ und $E \equiv T_{\overline{v}=\overline{V}}[\![u = N]\!]$.
2. $\mathcal{FV}(N) \subseteq \mathcal{V}(\overline{v})$ und für alle $1 \leq i \leq n$ gilt $\mathcal{FV}(V_i) \subseteq \{v_{i+1}, \ldots, v_n\}$.
3. Die Folge $u\overline{v}$ ist linear und die Mengen $\mathcal{V}(u\overline{v})$ und $\mathcal{BV}(N\overline{V})$ sind disjunkt.

In diesem Fall sind alle Substitutionen in $N\langle \overline{V}/\overline{v} \rangle$ konsistent, der Ausdruck $N\langle \overline{V}/\overline{v} \rangle$ geschlossen und u die einzige freie Variable in $T_{\overline{v}=\overline{V}}[\![u = N]\!]$.

Definition 4.4.1 *Für alle u definieren wir die Relation $S_u \subseteq \Lambda^{cl} \times \Delta^{ad}$ als Menge aller Paare (M, E) für die eine u-Darstellung $(N, \overline{V}, \overline{v})$ existiert.*

Proposition 4.4.2 *Für alle u ist S_u eine Längensimulation von Φ_u, wobei wir Φ_u als Einbettung von $(\Lambda^{cl}, \equiv, \rightarrow_e)$ nach $(\Delta^{ad}, \equiv, \stackrel{E}{\rightarrow}_A)$ auffassen.*

Beweisanfang. S_u enthält alle Paare der Form $(M, u = M)$ mit $M \in \Lambda^{cl}$, wobei wir für $\overline{v}$ und $\overline{V}$ die leeren Sequenzen einsetzen.

Die erste Eigenschaft (*LS1*) einer Längensimulation ist einfach nachzuweisen. Ist M irreduzibel in $\rightarrow_e$, dann ist M eine Abstraktion, da M als geschlossen vorausgesetzt ist. In diesem Fall ist $u = M$ eine Abstraktion; desweiteren ist $\overline{v} = \overline{V}$ eine Folge von Abstraktionen. Dann sind alle E mit $(M, E) \in S_u$ irreduzibel bezüglich $(\rightarrow_E \cup \rightarrow_A)$, also auch in $\stackrel{E}{\rightarrow}_A$. Denn Applikation (A) läßt sich nicht auf eine Komposition von Abstraktionen anwenden und Elimination (E) müßte die Gleichung $u = v$ verwenden, aber v ist global. Dies läßt sich formal durch Übergang zu Normalformen zeigen, also mit Hilfe von Satz 3.4.4 und Proposition 2.8.9.

Ohne weitere Vorbereitungen können wir nun Eigenschaft (*LS2*) einer Längensimulation für S_u noch nicht nachweisen. Diese formulieren wir in der folgenden Proposition 4.4.3 explizit. □

Zur Vervollständigung des Beweises von Proposition 4.4.2 müssen wir noch die nachstehende Proposition nachweisen, was wir einigen Vorbereitungen bewerkstelligen werden.

Proposition 4.4.3 *Für alle u, M, M', E mit $(M, E) \in S_u$ und $M \rightarrow_e M'$ existiert E' mit $(M', E') \in S_u$ und $E \stackrel{E}{\rightarrow}_A E'$.*

Als erstes Hilfsmittel benötigen wir eine Eigenschaft des sequentiellen Substitutionsoperators. Dazu sei $(N, \overline{V}, \overline{v})$ eine u-Darstellung, $\overline{v} = (v_1, \ldots, v_n)$ und $\overline{V} = (V_1, \ldots, V_n)$. Für $z \in \mathcal{V}(\overline{v})$ bezeichnen wir mit V_z die eindeutig bestimmte Abstraktion V_i mit $v_i = z$ und $1 \leq i \leq n$.

Lemma 4.4.4 *Ist $(N, \overline{V}, \overline{v})$ eine u-Darstellung, $z \in \mathcal{V}(\overline{v})$, und y eine beliebige Variable, dann gilt $N[z/y]\langle \overline{V} / \overline{v} \rangle = N[V_z/y]\langle \overline{V} / \overline{v} \rangle$.*

Beweis. Nach Definition einer Darstellung gilt $\mathcal{FV}(V_i) \subseteq \{v_{i+1}, \ldots, v_n\}$ für alle i. Daraus erhalten wir für beliebige i und M:

$$M[V_i/v_i]\langle \overline{V} / \overline{v} \rangle = M\langle \overline{V} / \overline{v} \rangle ,$$

denn die Substitutionen $[V_1/v_1] \ldots [V_i/v_i]$ haben keinen Effekt auf V_i. Durch zweimalige Anwendung dieser Eigenschaft folgt das Lemma:

$$\begin{aligned} N[z/y]\langle \overline{V}/\overline{v}\rangle &= N[z/y][V_z/v_z]\langle \overline{V}/\overline{v}\rangle \\ &= N[V_z/y][V_z/v_z]\langle \overline{V}/\overline{v}\rangle \\ &= N[V_z/y]\langle \overline{V}/\overline{v}\rangle \end{aligned}$$

□

Als nächstes benötigen wir Eigenschaften von Reduktion in Kontexten $T_{\overline{v}=\overline{V}}$. Diese hängen insbesondere davon ab, daß keine der in $T_{\overline{v}=\overline{V}}$ deklarierten Variablen, also keine Variablen aus $\overline{v}$ eliminiert wird. Diese Notwendigkeit erfassen wir formal, indem wir eine Relation $\rightarrow_E^{\overline{v}=\overline{V}}$ definieren, die Elimination im Kontext von $T_{\overline{v}=\overline{V}}$ geeignet einschränkt.

Als leichte Verallgemeinerung von $\overline{v} = \overline{V}$ betrachten wir nun eine beliebige Komposition von benannten Abstraktionen $\overline{v}\colon \overline{A}$ und definieren die Relation $\rightarrow_E^{\overline{v}:\overline{A}}$ wie folgt:

$$\exists \overline{w} \exists x (x = y \wedge F) \;\rightarrow_E^{\overline{v}:\overline{A}}\; \exists \overline{w} F[y/x] \qquad \begin{array}{l}\text{falls } x \neq y,\ y \in \mathcal{V}(\overline{v}\overline{w}) \\ \text{und } y \notin \mathcal{BV}(F)\end{array}$$

$$\frac{E_1 \equiv E_2 \qquad E_2 \rightarrow_E^{\overline{v}:\overline{A}} F_2 \qquad F_2 \equiv F_1}{E_1 \rightarrow_E^{\overline{v}:\overline{A}} F_1}$$

Die Deklarationen $\exists \overline{w}$ sind notwendig damit die Relation $\rightarrow_E^{\overline{v}:\overline{A}}$ abgeschlossen unter Deklaration ist.

Lemma 4.4.5 *Die Relation $\rightarrow_E^{\overline{v}:\overline{A}}$ ist invariant unter Komposition, Deklaration und Kongruenz.*

Beweis. Mit Induktion über Herleitungen von $\rightarrow_E^{\overline{v}:\overline{A}}$. Für den Nachweis der Invarianz unter Komposition benötigen wir die Einschränkung im Vergleich zur Elimination im Kontext $T_{\overline{v}:\overline{A}}$. □

Lemma 4.4.6 *Wenn $E \rightarrow_E^{\overline{v}:\overline{A}} F$, dann gilt $T_{\overline{v}:\overline{A}}[\![E]\!] \rightarrow_E T_{\overline{v}:\overline{A}}[\![F]\!]$.*

Beweis. Mit Induktion über Herleitungen von $\rightarrow_E^{\overline{v}:\overline{A}}$. □

Zur Applikation im Kontext $\overline{v}\colon \overline{A}$ definieren wir $\rightarrow_A^{\overline{v}:\overline{A}}$ ohne spezielle Einschränkungen:

$$E \;\rightarrow_A^{\overline{v}:\overline{A}}\; F \qquad gdw. \qquad T_{\overline{v}:\overline{A}}[\![E]\!] \;\rightarrow_A\; T_{\overline{v}:\overline{A}}[\![F]\!] .$$

Dann gilt die folgende Aussage in Analogie zu Lemma 4.4.5 mit Applikation anstelle von Elimination.

Lemma 4.4.7 *Die Relation* $\rightarrow_A^{\overline{v}:\overline{A}}$ *ist invariant unter Komposition, Deklaration und Kongruenz.*

Beweis. Mit Induktion über Herleitungen von $\rightarrow_A^{\overline{v}:\overline{A}}$. □

Da wir eigentlich nur Applikationsschritte zählen wollen, benötigen wir ein Analogon zur Relation $\xrightarrow{E}_A = (\rightarrow_E^* \circ \rightarrow_A \circ \rightarrow_E^*)$ für Reduktion im Kontext von $T_{\overline{v}:\overline{A}}$. Zu diesem Zweck definieren wir $\xrightarrow{E}_A^{\overline{v}:\overline{A}}$ durch:

$$\xrightarrow{E}_A^{\overline{v}:\overline{A}} = ((\rightarrow_E^{\overline{v}:\overline{A}})^* \circ \rightarrow_A^{\overline{v}:\overline{A}} \circ (\rightarrow_E^{\overline{v}:\overline{A}})^*) .$$

Diese zusammengesetzte Relation ererbt die Eigenschaften ihrer Komponenten, die wie in den voranstehenden Lemmata notiert haben.

Lemma 4.4.8 *Die Relation* $\xrightarrow{E}_A^{\overline{v}:\overline{A}}$ *ist invariant unter Komposition, Deklaration und Kongruenz.*

Beweis. Durch Kombination von Lemma 4.4.5 und Lemma 4.4.7. □

Lemma 4.4.9 *Wenn* $F_1 \xrightarrow{E}_A^{\overline{v}:\overline{A}} F_2$ *, dann* $T_{\overline{v}:\overline{A}}[\![F_1]\!] \xrightarrow{E}_A T_{\overline{v}:\overline{A}}[\![F_2]\!]$.

Beweis. Die analoge Aussage für Elimination gilt wegen Lemma 4.4.6 und für Applikation per Definition. □

Wir formulieren und beweisen nun die zentrale Invariante, auf der unser Adäquatheitsbeweis basiert.

Lemma 4.4.10 (Die Invariante) *Sei* M *geschlossen,* $M \rightarrow_e M'$ *und* $(N, \overline{V}, \overline{v})$ *eine* u*-Darstellung von* $(M, T_{\overline{v}=\overline{V}}[\![u = N]\!])$*. Dann existiert ein Tripel* $(N', \overline{W}, \overline{w})$ *mit folgenden Eigenschaften:*

1. $u = N \xrightarrow{E}_A^{\overline{v}=\overline{V}} T_{\overline{w}=\overline{W}}[\![u = N']\!]$.
2. $M' \equiv N'\langle \overline{W} / \overline{w} \rangle \langle \overline{V} / \overline{v} \rangle$.
3. *Die Folge* $u\overline{w}\overline{v}$ *ist linear,* $\mathcal{FV}(N') \subseteq \mathcal{V}(\overline{w}\overline{v})$ *und* $\mathcal{FV}(\overline{W}) \subseteq \mathcal{V}(\overline{v})$.

Bevor wir die Invariante nachweisen, zeigen wir, daß diese die benötigte zweite Eigenschaft einer Längensimulation für Φ_u impliziert.

Beweis von Proposition 4.4.3. Gelte $(M, E) \in S_u$ und $M \rightarrow_e M'$. Dann sind M und M' geschlossen und es existiert eine u-Darstellung $(N, \overline{V}, \overline{v})$ von $(M, T_{\overline{v}=\overline{V}}[\![u = N]\!])$, so daß wir Lemma 4.4.10 anwenden können. Dieses sichert die Existenz eines Tripels $(N', \overline{W}, \overline{w})$ mit folgenden Eigenschaften zu:

1. $u = N \overset{E}{\to}_A^{\overline{v}=\overline{V}} T_{\overline{w}=\overline{W}}[\![u = N']\!]$.
2. $M' \equiv N'\langle \overline{W} / \overline{w} \rangle \langle \overline{V} / \overline{v} \rangle$.
3. Die Folge $u\overline{w}\overline{v}$ ist linear, $\mathcal{FV}(N') \subseteq \mathcal{V}(\overline{w}\overline{v})$ und $\mathcal{FV}(\overline{W}) \subseteq \mathcal{V}(\overline{v})$.

Durch geeignete Umbennung von gebundenen Variablen in $\overline{W}$ finden wir $\overline{W'}$, so daß $(N', \overline{w}\overline{v}, \overline{W'}\overline{V})$ eine u-Darstellung von (M', E') mit $E' = T_{\overline{v}\overline{w}=\overline{V}\,\overline{W}}[\![u = N']\!]$ ist, und somit $(M', E') \in S_u$ gilt.

Letztendlich müssen wir nachweisen, daß $E \overset{E}{\to}_A E'$ erfüllt ist. Dies folgt aus der ersten der obigen Eigenschaften in Kombination mit den Lemmata 4.4.9 und 4.4.8:

$$E \equiv T_{\overline{v}=\overline{V}}[\![u = N]\!] \overset{E}{\to}_A T_{\overline{v}=\overline{V}}[\![T_{\overline{w}=\overline{W}}[\![u = N']\!]]\!] \equiv E' .$$

□

Beweis von Lemma 4.4.10. Mit Induktion über Herleitungen von $M \to_e M'$. Wir können ausschließen, daß N eine Variable oder eine Abstraktion ist, denn anderenfalls ist M eine Abstraktion und somit irreduzibel. Also sind N und M Applikationen der Form $N = N_1N_2$, $M = M_1M_2$, $M_1 = N_1\langle \overline{V} / \overline{v} \rangle$ und $M_2 = N_2\langle \overline{V} / \overline{v} \rangle$. Insbesondere ist $(N_1, \overline{V}, \overline{v})$ eine u-Darstellung von $(M_1, T_{\overline{v}=\overline{V}}[\![u = N_1]\!])$ und $(N_2, \overline{V}, \overline{v})$ eine u-Darstellung von $(M_2, T_{\overline{v}=\overline{V}}[\![u = N_2]\!])$.

1. **Fall** Die betrachtete Herleitung von $M \to_e M'$ stammt von (*Congr*) ab:

$$\frac{M \equiv \tilde{M} \qquad \tilde{M} \to_e \tilde{M}' \qquad \tilde{M}' \equiv M'}{M \to_e M'}$$

   Die Behauptung folgt nun aus der Induktionsvoraussetzung angewendet auf $\tilde{M} \to_e \tilde{M}'$, der Invarianz von $\to_e$ unter Kongruenz und der Definition einer u-Darstellung.

2. **Fall** Die letzte Regel der Herleitung von $M \to_e M'$ ist (*Func*):

$$\frac{M_1 \to_e M_1'}{M = M_1M_2 \to_e M_1'M_2 = M'}$$

   Ist u_1 eine frische Variable, dann existiert nach Induktionsvoraussetzung ein Tripel $(N_1', \overline{W}, \overline{w})$ mit folgenden Eigenschaften:

   (a) $u_1 = N_1 \overset{E}{\to}_A^{\overline{v}=\overline{V}} T_{\overline{w}=\overline{W}}[\![u_1 = N_1']\!]$.
   (b) $M_1' \equiv N_1'\langle \overline{W} / \overline{w} \rangle \langle \overline{V} / \overline{v} \rangle$.

(c) Die Folge $u_1\overline{w}\overline{v}$ ist linear, $\mathcal{FV}(N_1') \subseteq \mathcal{V}(\overline{w}\overline{v})$ und $\mathcal{FV}(\overline{W}) \subseteq \mathcal{V}(\overline{v})$.

Wir können ohne Einschränkung $u \notin \mathcal{V}(\overline{w})$ annehmen, was sich anderenfalls durch α-Umbenenung von Variablen in $\overline{w}$ erreichbar ist. Wir setzen nun $N' = N_1'N_2$ und zeigen das Lemma mit $(N', \overline{W}, \overline{w})$.

Nach Definition der Einbettung gilt mit einer weiteren frischen Variablen u_2

$$u = N \quad \equiv \quad \exists u_1 \exists u_2 (u_1 = N_1 \wedge u_2 = N_2 \wedge u_1\, u_2\, u)$$

Die 1. Eigenschaft des Lemmas können wir aus (2a) und Lemma 4.4.8 ableiten:

$$\begin{array}{rcl} u = N & \stackrel{E}{\to}_A^{\overline{v}=\overline{V}} & \exists u_1 \exists u_2 (\exists \overline{w}(u_1 = N_1' \wedge \overline{w} = \overline{W}) \wedge u_2 = N_2 \wedge u_1\, u_2\, u) \\ & \equiv & \exists \overline{w}(\exists u_1 \exists u_2 (u_1 = N_1' \wedge u_2 = N_2 \wedge u_1\, u_2\, u) \wedge \overline{w} = \overline{W}) \\ & \equiv & \exists \overline{w}(u = N_1'N_2 \wedge \overline{w} = \overline{W}) \\ & = & T_{\overline{w}=\overline{W}}[\![u = N']\!] \end{array}$$

Eigenschaft 2. im Lemma erhalten wir aus (2b) und der Disjunktheit von $V(\overline{w})$ und $\mathcal{FV}(N_2)$:

$$\begin{array}{rcl} M' = M_1'M_2 & \equiv & (N_1'\langle \overline{W} / \overline{w} \rangle \langle \overline{V} / \overline{v} \rangle)(N_2\langle \overline{V} / \overline{v} \rangle) \\ & = & (N_1'N_2)\langle \overline{W} / \overline{w} \rangle \langle \overline{V} / \overline{v} \rangle \\ & = & N'\langle \overline{W} / \overline{w} \rangle \langle \overline{V} / \overline{v} \rangle \end{array}$$

Die 3. Eigenschaft des Lemmas ist eine Konsequenz aus (2c) und $u \notin \mathcal{V}(\overline{w})$.

3. **Fall** Ist (Arg) die zuletzt angewendete Regel, dann können wir analog zum vorangehenden Fall schließen.

4. **Fall** Es verbleibt die Betrachtung des Axioms (β_e), also der Fall

$$M = M_1M_2 = (\lambda y_1.\tilde{M}_1)\, M_2 \;\to_{\beta e}\; \tilde{M}_1[M_2/y_1] = M'\,,$$

wobei M_2 eine Abstraktion ist und die Mengen $\mathcal{FV}(M_2)$ und $\mathcal{BV}(\tilde{M}_1)$ disjunkt sind.

(a) Seien nun N_1 und N_2 Variablen, etwa $N_1 = z_1$ und $N_2 = z_2$. Dann gilt $M_1 = z_1\langle \overline{V} / \overline{v} \rangle$ und $M_2 = z_2\langle \overline{V} / \overline{v} \rangle$. Wegen $\{z_1, z_2\} \subseteq \mathcal{V}(\overline{v})$ existieren V_{z_1} und V_{z_2} mit der Eigenschaft von Lemma 4.4.4, so daß M_1 die folgende Darstellung besitzt:

$$M_1 \;=\; z_1\langle \overline{V} / \overline{v} \rangle \;=\; z_1[z_1/z_1]\langle \overline{V} / \overline{v} \rangle \;=\; z_1[V_{z_1}/z_1]\langle \overline{V} / \overline{v} \rangle \;=\; V_{z_1}\langle \overline{V} / \overline{v} \rangle$$

Andererseits wissen wir $M_1 \equiv \lambda y_1.\tilde{M}_1$ und somit existiert $\tilde{N}_1$, so daß $V_{z_1} \equiv \lambda y_1.\tilde{N}_1$ und $\tilde{M}_1 \equiv \tilde{N}_1\langle\overline{V}/\overline{v}\rangle$. Nach Definition der Einbettung gilt mit frischen Variablen y_1', u_1 und u_2:

$$\begin{array}{rcl} z_1 = V_{z_1} & \equiv & z_1{:}y_1\,y_1'/y_1' = \tilde{N}_1\,, \\ u = N_1N_2 & \equiv & \exists u_1\exists u_2(u_1 = z_1 \wedge u_2 = z_2 \wedge u_1\,u_2\,u)\,. \end{array}$$

Dies ermöglich die folgende Ableitung:

$$u = N_1N_2 \;(\rightarrow_E^{\overline{v}=\overline{V}})^*\; z_1\,z_2\,u \;\rightarrow_A^{\overline{v}=\overline{V}}\; u = \tilde{N}_1[z_2/y_1]\,.$$

Nun definieren wir $N' = \tilde{N}_1[z_2/y_1]$ und $\overline{w}$ und $\overline{W}$ als leere Sequenzen. Dann erfüllt das Tripel $(N', \overline{W}, \overline{w})$ die Eigenschaften des Lemmas. 1. haben wir bereits gezeigt, 3. ist trivial und 2. folgt aus $y_1 \notin \mathcal{V}(\overline{v})$, was wegen $y_1 \in \mathcal{BV}(\overline{V})$ gilt:

$$\begin{array}{rcl} N'\langle\overline{w}/\overline{W}\rangle\langle\overline{V}/\overline{v}\rangle & = & \tilde{N}_1[z_2/y_1]\langle\overline{V}/\overline{v}\rangle \\ & \equiv & \tilde{N}_1\langle\overline{V}/\overline{v}\rangle[z_2\langle\overline{V}/\overline{v}\rangle/y_1] \\ & \equiv & \tilde{M}_1[M_2/y_1] \\ & \equiv & M' \end{array}$$

Die folgenden Fälle sind alle analog zu dem hiermit gezeigten.

(b) Sei nun N_1 eine Abstraktion und N_2 eine Variable. Dann gilt $M_1 = N_1\langle\overline{V}/\overline{v}\rangle$ und es gibt eine Variable z_2 mit $z_2 = N_2$ und $M_2 = z_2\langle\overline{V}/\overline{v}\rangle$. Wegen $z_2 \in \mathcal{V}(\overline{v})$ existiert V_{z_2} mit der Eigenschaft von Lemma 4.4.4, so daß M_2 die folgende Darstellung besitzt:

$$M_2 \;=\; z_1\langle\overline{V}/\overline{v}\rangle \;=\; z_1[z_2/z_2]\langle\overline{V}/\overline{v}\rangle \;=\; z_1[V_{z_2}/z_2]\langle\overline{V}/\overline{v}\rangle \;=\; V_{z_2}\langle\overline{V}/\overline{v}\rangle$$

Ist $M_1 = \lambda y_1.\tilde{M}_1$, dann existiert $\tilde{N}_1$ mit $N_1 = \lambda y_1.\tilde{N}_1$ und $\tilde{M}_1 \equiv \tilde{N}_1\langle\overline{V}/\overline{v}\rangle$. Nach Definition der Einbettung gilt mit frischen Variablen y_1', u_1 und u_2:

$$\begin{array}{rcl} u_1 = N_1 & \equiv & u_1{:}y_1\,y_1'/y_1' = \tilde{N}_1\,, \\ u = N_1N_2 & \equiv & \exists u_1\exists u_2(u_1 = N_1 \wedge u_2 = z_2 \wedge u_1\,u_2\,u)\,. \end{array}$$

Wir können wie folgt reduzieren:

$$\begin{array}{rcl} u = N_1N_2 & \rightarrow_E^{\overline{v}=\overline{V}} & \exists u_1(u_1 = N_1 \wedge u_1\,z_2\,u) \\ & \rightarrow_A^{\overline{v}=\overline{V}} & \exists u_1(u_1 = N_1 \wedge u = \tilde{N}_1[z_2/y_1]) \end{array}$$

Wir definieren nun $N' = \tilde{N}_1[z_2/y_1]$, sowie $\overline{W} = (N_1)$ und $\overline{w} = (u_1)$. Dann erfüllt das Tripel $(N', \overline{W}, \overline{w})$ die behaupteten Eigenschaften. Die

1. haben wir bereits gezeigt, die 3. ist trivial und die 2. folgt aus $y_1 \notin \mathcal{V}(\overline{v})$, was wegen $y_1 \in \mathcal{BV}(\overline{V})$ gilt:

$$\begin{array}{rcl} N'\langle\overline{w}/\overline{W}\rangle\langle\overline{V}/\overline{v}\rangle & = & \tilde{N}_1[z_2/y_1][N_1/u_1]\langle\overline{V}/\overline{v}\rangle \\ & = & \tilde{N}_1[z_2/y_1]\langle\overline{V}/\overline{v}\rangle \qquad \text{da } u_1 \notin \mathcal{FV}(\tilde{N}_1 z_2) \\ & \equiv & \tilde{N}_1\langle\overline{V}/\overline{v}\rangle[z_2\langle\overline{V}/\overline{v}\rangle/y_1] \\ & \equiv & \tilde{M}_1[M_2/y_1] \\ & \equiv & M' \end{array}$$

(c) Jetzt betrachten wir den Fall, daß N_1 ist eine Variable und N_2 eine Abstraktion ist. Dann gilt $M_2 = N_2\langle\overline{V}/\overline{v}\rangle$ und es gibt eine Variable z_1 mit $N_1 = z_1$ und $M_1 = z_1\langle\overline{V}/\overline{v}\rangle$. Wegen $z_1 \in \mathcal{V}(\overline{v})$ existiert die Abstraktion V_{z_1} mit den Eigenschaften von Lemma 4.4.4, so daß M_1 die folgende Darstellung besitzt:

$$M_1 \;=\; z_1\langle\overline{V}/\overline{v}\rangle \;=\; z_1[z_1/z_1]\langle\overline{V}/\overline{v}\rangle \;=\; z_1[V_{z_1}/z_1]\langle\overline{V}/\overline{v}\rangle \;=\; V_{z_1}\langle\overline{V}/\overline{v}\rangle$$

Ist $M_1 = \lambda y_1.\tilde{M}_1$, dann existiert $\tilde{N}_1$ mit $V_{z_1} = \lambda y_1.\tilde{N}_1$ und $\tilde{M}_1 = \tilde{N}_1\langle\overline{V}/\overline{v}\rangle$. Nach Definition der Einbettung gilt mit frischen Variablen y_1', u_1 und u_2:

$$\begin{array}{rcl} z_1 = V_{z_1} & \equiv & z_1{:}y_1\, y_1'/y_1' = \tilde{N}_1\,, \\ u = N_1 N_2 & \equiv & \exists u_1 \exists u_2\,(u_1 = z_1 \wedge u_2 = N_2 \wedge u_1\, u_2\, u)\,. \end{array}$$

Wir können nun wie folgt reduzieren:

$$\begin{array}{rcl} u = N_1 N_2 & \to_E^{\overline{v}=\overline{V}} & \exists u_2\,(u_2 = N_2 \wedge z_1\, u_2\, u) \\ & \to_A^{\overline{v}=\overline{V}} & \exists u_2\,(u_2 = N_2 \wedge u = \tilde{N}_1[u_2/y_1]) \end{array}$$

Wir definieren nun $N' = \tilde{N}_1[u_2/y_1]$, sowie $\overline{w} = (u_2)$ und $\overline{W} = (N_2)$. Dann erfüllt das Tripel $(N', \overline{W}, \overline{w})$ die behaupteten Eigenschaften: Die 1. haben wir bereits gezeigt, die 3. ist trivial und die 2. folgt aus $y_1 \notin \mathcal{V}(\overline{v})$, was wegen $y_1 \in \mathcal{BV}(\overline{V})$ gilt:

$$\begin{array}{rcl} N'\langle\overline{w}/\overline{W}\rangle\langle\overline{V}/\overline{v}\rangle & = & \tilde{N}_1[u_2/y_1][N_2/u_2]\langle\overline{V}/\overline{v}\rangle \\ & = & \tilde{N}_1[N_2/y_1]\langle\overline{V}/\overline{v}\rangle \qquad \text{da } u_2 \notin \tilde{N}_1 \\ & \equiv & \tilde{N}_1\langle\overline{V}/\overline{v}\rangle[N_2\langle\overline{V}/\overline{v}\rangle/y_1] \\ & \equiv & \tilde{M}_1[M_2/y_1] \\ & \equiv & M' \end{array}$$

(d) Seien N_1 und N_2 Abstraktionen. Dann gilt $M_1 = N_1\langle\overline{V}/\overline{v}\rangle$ und $M_2 = N_2\langle\overline{V}/\overline{v}\rangle$. Ist $M_1 = \lambda y_1.\tilde{M}_1$, dann existiert y_1 und $\tilde{N}_1$ mit $N_1 = \lambda y_1.\tilde{N}_1$

und $\tilde{M}_1 = \tilde{N}_1\langle\overline{V}/\overline{v}\rangle$. Nach Definition der Einbettung gilt mit frischen Variablen y_1', u_1 und u_2:

$$\begin{array}{rcl} z_1 = N_1 & \equiv & z_1{:}y_1\, y_1'/y_1' = \tilde{N}_1\ , \\ u = N_1 N_2 & \equiv & \exists u_1 \exists u_2 (u_1 = N_1 \wedge u_2 = N_2 \wedge u_1\, u_2\, u)\ . \end{array}$$

Nun können wir folgendermaßen reduzieren:

$$u = N_1 N_2 \quad \to_A^{\overline{v}=\overline{V}} \quad \exists u_1 \exists u_2 (u_1 = N_1 \wedge u_2 = N_2 \wedge u = \tilde{N}_1[u_2/y_1])$$

Wir definieren $N' = \tilde{N}_1[u_2/y_1]$, sowie $\overline{w} = (u_1, u_2)$ und $\overline{W} = (N_1, N_2)$. Dann erfüllt das Tripel $(N', \overline{W}, \overline{w})$ die Eigenschaften des Lemmas: Die 1. haben wir bereits gezeigt, die 3. ist trivial und die 2. folgt aus $y_1 \notin \mathcal{V}(\overline{v})$, was wegen $y_1 \in \mathcal{BV}(\overline{V})$ gilt, und wegen $u_1 \neq u_2$ und $u_1, u_2 \notin \tilde{N}_1$:

$$\begin{array}{rcl} N'\langle\overline{w}/\overline{W}\rangle\langle\overline{V}/\overline{v}\rangle & = & \tilde{N}_1[u_2/y_1][N_1/u_1][N_2/u_2]\langle\overline{V}/\overline{v}\rangle \\ & = & \tilde{N}_1[N_2/y_1]\langle\overline{V}/\overline{v}\rangle \\ & \equiv & \tilde{N}_1\langle\overline{V}/\overline{v}\rangle[N_2\langle\overline{V}/\overline{v}\rangle/y_1] \\ & \equiv & \tilde{M}_1[M_2/y_1] \\ & \equiv & M' \end{array}$$

□

Kapitel 5

Basen des δ-Kalküls

In Abschnitt 3.4 haben wir Definitions- und Reduktionsbasen für den δ-Kalkül angegeben, die wir zum Beweis von uniformer Konfluenz ausgenützt haben. In diesem Abschnitt werden wir nun die Korrektheit der angegeben Basen beweisen. Wir beginnen mit einer Zusammenstellung von Notationen und betrachten anschließend α-standardisierte Ausdrücke und Normalformen. Besonderen Wert legen wir auf die Vollständigkeit unserer Definitionen und Beweise.

5.1 Notationen

Um die Lesbarkeit zu erhöhen geben wir zuerst einen Überblick über Notationen für binäre Relationen auf Δ. Die meisten von diesen haben wir bereits in Kapitel 3 eingeführt.

Sei R eine beliebige binäre Relation auf Δ. Die Relation $\to_R$ ist die kleinste Relation, die R enthält und die Regeln $(Decl)$, $(Comp)$ und $(Congr)$ erfüllt. Wir erinneren daran, daß diese Regeln die folgende Gestalt haben:

$$(Decl)\quad \frac{E \to F}{\exists x\, E \to \exists x\, F} \qquad (Abstr)\quad \frac{E \to F}{x{:}\overline{y}/E \to x{:}\overline{y}/F}$$

$$(Congr)\quad \frac{E_1 \equiv E_2 \qquad E_2 \to F_2 \qquad F_2 \equiv F_1}{E_1 \to F_1}$$

$$(Comp)\quad \frac{E \to F}{E \wedge G \to F \wedge G} \qquad (Comp')\quad \frac{E \;\to\; F}{G \wedge E \;\to\; G \wedge F}$$

Die Relation $\equiv_R$ ist die kleinste Kongruenz, die R enthält, also die kleinste Äquivalenzrelation, die R enthält und die Regeln $(Decl)$, $(Comp)$, $(Comp')$ und $(Abstr)$ erfüllt. Die folgenden Abkürzungen für spezielle Kongruenzen sind bereits aus Ka-

pitel 3 bekannt:

$$\equiv \;=\; \equiv_{\{\alpha,Scope,Exch,ACI\}}\,, \qquad \equiv_1 \;=\; \equiv_{\{Scope,Exch,ACI\}}\,, \qquad \equiv_2 \;=\; \equiv_{\{Exch,ACI\}}\,.$$

Die Relation $\bar{\to}_R$ ist die kleinste binäre Relation auf Δ, die R enthält und die Regeln $(Decl)$, $(Comp)$ und $(Comp')$ erfüllt. Die Relation $\Rightarrow_R$ ist die kleinste binäre Relation auf Δ, die R enthält und die Regeln $(Decl)$, $(Comp)$, $(Comp')$ und $(Abstr)$ erfüllt. Aus Symmetriegründen bezeichnen wir mit $\bar{\to}_R$ die Relation R selbst.

Proposition 5.1.1 *Seien R_1 und R_2 zwei binäre Relationen auf Δ. Dann gilt*

$$\begin{array}{lcllcl} \equiv_{R_1\cup R_2} & = & (\equiv_{R_1} \cup \equiv_{R_2})\,, & \Rightarrow_{R_1\cup R_2} & = & (\Rightarrow_{R_1} \cup \Rightarrow_{R_2})\,, \\ \bar{\to}_{R_1\cup R_2} & = & (\bar{\to}_{R_1} \cup \bar{\to}_{R_2})\,, & \bar{\to}_{R_1\cup R_2} & = & (\bar{\to}_{R_1} \cup \bar{\to}_{R_2})\,. \end{array}$$

Beweis. Wir führen den Beweis nur für $\bar{\to}_{R_1\cup R_2} = (\bar{\to}_{R_1} \cup \bar{\to}_{R_2})$. Die anderen Fälle sind analog. Die links stehende Relation $\to_{R_1\cup R_2}$ enthält R_1 (bzw. R_2) und erfüllt $(Decl)$, $(Comp)$ und $(Comp')$. Da $\bar{\to}_{R_1}$ (bzw. $\bar{\to}_{R_2}$) minimal mit diesen Eigenschaften ist, folgt:

$$\bar{\to}_{R_1\cup R_2} \supseteq \bar{\to}_{R_1} \qquad \text{und} \qquad \bar{\to}_{R_1\cup R_2} \supseteq \bar{\to}_{R_1}\,,$$

also auch $\bar{\to}_{R_1\cup R_2} \supseteq (\bar{\to}_{R_1} \cup \bar{\to}_{R_2})$. Die rechts stehende Relation $(\bar{\to}_{R_1} \cup \bar{\to}_{R_2})$ enthält $R_1 \cup R_2$ und erfüllt $(Decl)$, $(Comp)$ und $(Comp')$, wie sich leicht nachrechnen läßt. Nun ist $\bar{\to}_{R_1\cup R_2}$ die kleinste Relation mit diesen Eigenschaften, woraus die Inklusion $\bar{\to}_{R_1\cup R_2} \subseteq (\bar{\to}_{R_1} \cup \bar{\to}_{R_2})$ folgt. □

Proposition 5.1.2 *Für jede binäre Relation R auf Δ gilt $\to_R = (\equiv \circ \bar{\to}_R \circ \equiv)$.*

Beweis. Ähnlich zum Beweis von Proposition 5.1.1. Die Implikation $\to_R \supseteq (\equiv \circ \bar{\to}_R \circ \equiv)$ ist trivial. Die umgekehrte Implikation folgt aus der Minimalität von $\to_R$. □

5.2 Basen durch α-Standardisierung

Wir beweisen nun die Korrektheit der Sätze 3.4.1 und 3.4.2. Diese geben Definitions- und Reduktionsbasen des δ-Kalküls basierend auf α-Standardisierung an.

5.2.1 α-Standarisierung

Ein Ausdruck E heißt α-standardisiert, falls für keine Variable mehrere Variablenbinder in E existieren und keine Variable in E sowohl freie als auch gebundene Aufteten besitzt. Formal definieren wir die Menge Δ^α aller α-standarisierten Ausdrücke als kleinste Menge mit den Eigenschaften in Abbildung 5.1.

$$\top \in \Delta^\alpha \qquad x\,\overline{y} \in \Delta^\alpha \qquad x = y \in \Delta^\alpha$$

$$\frac{E \in \Delta^\alpha}{\overline{y}\,/\,E \in \Delta^\alpha} \text{ falls } \mathcal{V}(\overline{y}) \text{ und } \mathcal{BV}(E) \text{ disjunkt und } \overline{y} \text{ linear}$$

$$\frac{A \in \Delta^\alpha}{x\colon A \in \Delta^\alpha} \text{ falls } x \cap \mathcal{BV}(A) = \emptyset \qquad \frac{E \in \Delta^\alpha}{\exists x\, E \in \Delta^\alpha} \text{ falls } x \notin \mathcal{BV}(E)$$

$$\frac{E \in \Delta^\alpha \qquad F \in \Delta^\alpha}{E \wedge F \in \Delta^\alpha} \text{ falls } \mathcal{BV}(E) \text{ und } \mathcal{V}(F) \text{ sowie } \mathcal{BV}(F) \text{ und } \mathcal{V}(E) \text{ disjunkt}$$

Abbildung 5.1: α-Standardisierung

Proposition 5.2.1 *Für alle E existiert F in Δ^α mit $E \equiv F$. Also liegt Δ^α in Δ bezüglich $\equiv$ dicht.*

Beweis. Mit Induktion über die Struktur von E. □

Wir notieren zwei einfache, aber wichtige Eigenschaften im Zusammenhang mit α-Standardisierung[1].

Lemma 5.2.2 *Im Fall $E \equiv_1 F$ haben E und F dieselben freien und gebundenen Variablen, also $\mathcal{BV}(E) = \mathcal{BV}(F)$ und $\mathcal{FV}(E) = \mathcal{FV}(F)$.*

Beweis. Mit Induktion über Herleitungen von $E \equiv_1 F$. □

Lemma 5.2.3 *$\equiv_1$ erhält α-Standardisierung.*

Beweis. Wir müssen unter der Voraussetzung $\tilde{E} \in \Delta^\alpha$ und $\tilde{E} \equiv_1 \tilde{F}$ zeigen, daß $\tilde{F} \in \Delta^\alpha$ gilt. Dies können wir mit Induktion über Herleitungen von $\tilde{E} \equiv_1 \tilde{F}$ nachweisen, was ziemlich langweilig ist.

Für die Regel (*Comp*) nehmen wir $\tilde{E} = E \wedge G$ und $\tilde{F} = F \wedge G$ und $\tilde{E} \in \Delta^\alpha$ an.

$$\frac{E \;\equiv_1\; F}{E \wedge G \;\equiv_1\; F \wedge G}$$

[1] Beide Eigenschaften würden verloren gehen, wenn wir wie im π-Kalkül $\exists x \top \equiv \top$ zulassen würden [Mil91].

Aus $\tilde{E} \in \Delta^\alpha$ folgt $E \in \Delta^\alpha$ und somit gilt $F \in \Delta^\alpha$ nach Induktionsannahme. Wir haben also bisher die folgenden Eigenschaften nachgewiesen:

$$1)\ E \wedge G \in \Delta^\alpha\,, \qquad 2)\ E \equiv_1 F\,, \qquad 3)\ F \in \Delta^\alpha\,.$$

Wir müssen nun $F \wedge G \in \Delta^\alpha$ zeigen, was äquivalent zur Gültigkeit der folgenden Eigenschaften ist:

$$4)\ F \in \Delta^\alpha\,, \qquad 5)\ G \in \Delta^\alpha\,, \qquad 6)\ \mathcal{BV}(F) \cap \mathcal{V}(G) = \emptyset\,, \qquad 7)\ \mathcal{BV}(G) \cap \mathcal{V}(F) = \emptyset\,.$$

Formel 3) ist gleich 4). Formel 1) impliziert 5) sowie die Eigenschaften:

$$6')\ \mathcal{BV}(E) \cap \mathcal{V}(G) = \emptyset \qquad \text{und} \qquad 7')\ \mathcal{BV}(G) \cap \mathcal{V}(E) = \emptyset.$$

Wegen Lemma 5.2.2 und 2) haben F und E die gleichen freien und gebundenen Variablen, so daß 6') und 7') äquivalent zu 6) bzw. 7) sind. □

5.2.2 Kontexte

Wir definieren Kontexte als erweiterte Ausdrücke, die möglicherweise ein Loch enthalten. Wir unterscheiden (starke) Kontexte S und schwache Kontexte T durch folgende Definition:

$$\begin{array}{rcl} S & ::= & \bullet \mid \exists x\, S \mid S \wedge E \mid E \wedge S \mid E \mid x{:}\overline{y}/S \\ T & ::= & \bullet \mid \exists x\, T \mid T \wedge E \mid E \wedge T \mid E \end{array}$$

Dabei ist $\bullet$ ein Platzhalter. Wir definieren Substitutionen $S_1[\![S_2]\!]$, die den Platzhalter $\bullet$ in S_1 durch S_2 ersetzen, homomorph über die Struktur von S_1:

$$\begin{array}{rclcrcl} \bullet[\![S_2]\!] & = & S_2 & & \exists x\, S_1[\![S_2]\!] & = & \exists x\, S_1[\![S_2]\!] \\ (S_1 \wedge E)[\![S_2]\!] & = & S_1[\![S_2]\!] \wedge E & & (E \wedge S_1)[\![S_2]\!] & = & E \wedge S_1[\![S_2]\!] \\ E[\![S_2]\!] & = & E & & (x{:}\overline{y}/S_1)[\![S_2]\!] & = & x{:}\overline{y}/S_1[\![S_2]\!] \end{array}$$

Proposition 5.2.4 *Sei R eine binäre Relation auf Δ.*

1. *Es gilt $E \Rightarrow_R F$ genau dann, wenn ein starker Kontext S und Ausdrücke E_1 und F_1 existieren mit $E = S[\![E_1]\!]$, $E_1 \stackrel{-}{\rightarrow}_R F_1$ und $F = S[\![F_1]\!]$.*

2. *Es gilt $E \stackrel{-}{\rightarrow}_R F$ genau dann, wenn ein schwacher Kontext T und Ausdrücke E_1 und F_1 existieren mit $E = T[\![E_1]\!]$, $E_1 \stackrel{-}{\rightarrow}_R F_1$ und $F = T[\![F_1]\!]$.*

Beweis. Mit Induktion über die Definition von starken und schwachen Kontexten S und T bzw. mit Induktion über Ableitungen von $E \Rightarrow_R F$ und $E \bar{\rightarrow}_R F$. □

Die meisten Begriffe und Notationen verallgemeinern sich von Ausdrücken auf Kontexte. Zum Beispiel können wir $\mathcal{FV}(\bullet) = \mathcal{BV}(\bullet) = \emptyset$ definieren. Etwas unangenehmer ist die Übertragung der binären Relationen $\rightarrow_R$, $\bar{\rightarrow}_R$ und $\Rightarrow_R$. Denn diese Relationen sind nicht abgeschlossen auf der Menge alle Kontexte, sonderen erzeugen erweiterte Kontexte der Form $\bullet[\overline{y}/\overline{x}]$. Wir wollen diesen Effekt aber nicht ausbuchstabieren, sondern werden frei mit erweiterten Kontexten rechnen.

Lemma 5.2.5 *Seien S_1, S_2, x und y beliebig. Dann gelten:*

1. $(S_1[\![S_2]\!])[y/x] \;=\; (S_1[y/x])[\![S_2[y/x]]\!]$.
2. $\mathcal{FV}(S_1[\![S_2]\!]) = \mathcal{FV}(S_1) \cup \mathcal{FV}(S_2)$.
3. $\mathcal{BV}(S_1[\![S_2]\!]) = \mathcal{BV}(S_1) \cup \mathcal{BV}(S_2)$.

Beweis. Mit Induktion über die Struktur von S_1. Wir betrachten hier nur die erste Eigenschaft im Fall $S_1 = \bullet$:

$$\begin{array}{rcl}(\bullet[y/x])[\![S_1[y/x]]\!] & = & S_1[y/x][y/x] \\ & = & S_1[y/x] \\ & = & (\bullet[\![S_1]\!])[y/x]\end{array}$$

□

Von hierab werden wir Kontexte und Ausdrücke nur falls notwendig explizit unterscheiden und beide mit den Metavariablen E, F und G denotieren. Wir erlauben also Kontexte oder Ausdrücke der Form $E_1[\![E_2]\!]$ oder $E_1[\![E_2[\![E_3]\!]]\!]$.

5.2.3 Die Relation $\stackrel{s}{\Rightarrow}_\alpha$ und Nachweis der Basen

Kontexte erlauben eine einfache und formale Definition der Relation $\stackrel{s}{\Rightarrow}_\alpha$. Diese Relation beschreibt diejenige Einschränkung von $\Rightarrow_\alpha$, durch deren Anwendung stets neue gebundene Variable eingeführt werden:

$$G \;=\; E[\![\exists x\, F]\!] \;\stackrel{s}{\Rightarrow}_\alpha\; E[\![\exists y\, F[y/x]]\!] \qquad \textit{falls } y \notin \mathcal{V}(G)$$

$$G \;=\; E[\![u\colon \overline{v} x A]\!] \;\stackrel{s}{\Rightarrow}_\alpha\; E[\![u\colon \overline{v} y A[y/x]]\!] \qquad \textit{falls } y \notin \mathcal{V}(G)$$

Nun formulieren wir von Eigenschaften von α-Umbennung, die hinreichend für alle Manipulationen in dieser Arbeit sind.

Proposition 5.2.6 (Die Details über α-Umbennung)

1. *Es gilt* $\equiv_\alpha \subseteq (\overset{s}{\Rightarrow}{}^*_\alpha \circ {}^*_\alpha\overset{s}{\Leftarrow})$.
2. *Aus* $E \in \Delta^\alpha$, $F \in \Delta$ *und* $E \overset{s}{\Rightarrow}_\alpha F$ *folgt* $E \ {}_\alpha\overset{s}{\Leftarrow} F$.
3. *Die Menge* Δ^α *ist abgeschlossen unter* $\overset{s}{\Rightarrow}_\alpha$.
4. *Die Relation* ${}^*_\alpha\overset{s}{\Leftarrow}$ *vertauscht mit* $\equiv_1$ *und* $\equiv_2$.

Den Beweis dieser Proposition werden wir in Abschnitt 5.2.4 führen. Als nächstes zeigen wir, daß sich aus Proposition 5.2.6 die Korrektheit der auf α-Standardisierung basierenden Definitionsbasen folgern läßt.

Beweis von Satz 3.4.1.

1. Die Relationen $\overset{s}{\Rightarrow}{}^*_\alpha$ und $\equiv_1$ vertauschen eingeschränkt auf $\Delta^\alpha \times \Delta^\alpha$:

   Nach Teil 4 von Proposition 5.2.6 vertauschen $\overset{s}{\Rightarrow}{}^*_\alpha$ und $\equiv_1$ auf $\Delta \times \Delta$. Da die Menge Δ^α abgeschlossen unter $\overset{s}{\Rightarrow}_\alpha$ und $\equiv_1$ ist (Proposition 5.2.6 Teil 3 und Lemma 5.2.3), folgt die Vertauschungseigenschaft auch eingeschränkt auf $\Delta^\alpha \times \Delta^\alpha$.

2. Eingeschränkt auf $\Delta^\alpha \times \Delta^\alpha$ gilt $\overset{s}{\Rightarrow}{}^*_\alpha \subseteq {}^*_\alpha\overset{s}{\Leftarrow}$:

   Dies folgt mit Induktion über die Anzahl der Schritte in $\overset{s}{\Rightarrow}{}^*_\alpha$. Dazu benötigen wir die Teile 2 und 3 von Proposition 5.2.6.

3. Das Paar $(\Delta^\alpha, (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ ist eine Definitionsbasis von $(\Delta, \equiv)$:

   Nach Proposition 5.2.1 liegt Δ^α dicht in Δ bezüglich $\equiv$. Wir müssen also zeigen, daß die Relationen $(\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*$ und $\equiv$ eingeschränkt auf $\Delta^\alpha \times \Delta^\alpha$ übereinstimmen. Die Inklusion $\equiv_{|\Delta^\alpha \times \Delta^\alpha} \supseteq (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*_{|\Delta^\alpha \times \Delta^\alpha}$ gilt offensichtlich. Der Trick zum Nachweis der umgekehrten Implikation besteht in der Verwendung der Identität:

   $$\equiv \ = \ (\equiv_1 \circ \equiv_\alpha)^* .$$

   Denn beide Relationen sind Kongruenzen, die alle Axiome von $\equiv$ enthalten, und minimal mit dieser Eigenschaft. Nach Definition der reflexiven transitiven Hülle reicht es nun aus,

   $$(\equiv_1 \circ \equiv_\alpha)^n_{|\Delta^\alpha \times \Delta^\alpha} \ \subseteq \ (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*_{|\Delta^\alpha \times \Delta^\alpha}$$

   für alle $n \geq 0$ nachzuweisen. Dies läßt sich mit Induktion über n und unter Verwendung von allen Teilen von Proposition 5.2.6 bewerkstelligen. Für den Induktionsschritt nehmen wir $E, F \in \Delta^\alpha$ und $n > 0$ mit

   $$E \ (\equiv_1 \circ \equiv_\alpha)^n \ F$$

an. Dann gibt es $G \in \Delta$ mit $E\ (\equiv_1 \circ \equiv_\alpha)\ G$ und $G\ (\equiv_1 \circ \equiv_\alpha)^{n-1}\ F$. Wegen Teil 1 von Proposition 5.2.6 existiert $G' \in \Delta$ mit

$$E\ (\equiv_1 \circ \overset{s}{\Rightarrow}{}^*_\alpha)\ G' \quad {}^*_\alpha\overset{s}{\Leftarrow}\ G\ .$$

Im Fall $n = 1$ gilt $G'\ {}^*_\alpha\overset{s}{\Leftarrow}\ G = F$. Wie bereits gezeigt folgt daraus $G' \overset{s}{\Rightarrow}{}^*_\alpha F$ und somit die Behauptung. Sei nun $n > 1$. Nach Teil 4 von Proposition 5.2.6 vertauschen $\equiv_1$ und $\overset{s}{\Rightarrow}_\alpha$, so daß wir netterweise

$$G'\ (\equiv_1 \circ \equiv_\alpha)^{n-1}\ F$$

schließen können, wozu wir $n > 1$ benötigen. Aus $E \in \Delta^\alpha$, Lemma 5.2.3 und Teil 3 von Proposition folgt $G' \in \Delta^\alpha$. Wegen $G', F \in \Delta^\alpha$ können wir nun die Induktionsvoraussetzung anwenden und erhalten $G'\,(\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*\ F$, also auch $E\ (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*\ F$. □

Nun verwenden wir die bewiesene Definitionsbasis aus Satz 3.4.1 zum Nachweis der Reduktionsbasis von Satz 3.4.2.

Proposition 5.2.7 *Aus $E \in \Delta^\alpha$, $F \in \Delta$ und $E \equiv F$ folgt $E\ (\equiv_1 \circ {}^*_\alpha\overset{s}{\Leftarrow})\ F$.*

Beweis. Seien $E \in \Delta^\alpha$ und $F \in \Delta$ mit $E \equiv F$. Wegen Proposition 5.2.1 existiert $G_1 \in \Delta^\alpha$ mit $E \equiv G_1 \equiv_\alpha F$. Nach Teil 1 von Proposition 5.2.6 existiert G_2 mit $G_1 \overset{s}{\Rightarrow}{}^*_\alpha\ G_2\ {}^*_\alpha\overset{s}{\Leftarrow}\ F$. Aus Teil 3 von Proposition 5.2.6 folgt $G_2 \in \Delta^\alpha$. Nach Satz 3.4.1 ist das Paar $(\Delta^\alpha, (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ eine Definitionsbasis von $(\Delta, \equiv)$, so daß wir $G_2\ (\equiv_1 \cup \overset{s}{\Rightarrow}_\alpha)^*\ E$ schließen können. Nun vertauschen $\equiv_1$ und $\overset{s}{\Rightarrow}_\alpha$ nach Teil 4 von Proposition 5.2.6, woraus $E\ (\equiv_1 \circ {}^*_\alpha\overset{s}{\Leftarrow})\ G_2$ folgt. Somit gilt die Behauptung $E\ (\equiv_1 \circ {}^*_\alpha\overset{s}{\Leftarrow})\ F$. □

Proposition 5.2.8 *Sei R eine der Relationen, die zu einem der Axiom (A), (E) oder (G) korrespondiert. Dann gilt* $({}^*_\alpha\overset{s}{\Leftarrow} \circ \bar{\to}_R)\ \subseteq\ (\bar{\to}_R \circ {}^*_\alpha\Leftarrow)$.

Der Beweis ist länglich und wird in Abschnitt 5.2.4 geführt.

Beweis von Satz 3.4.2. Sei R eine der Relationen (A), (E) oder (G). Wir müssen zeigen, daß $(\Delta^\alpha, \equiv_1, \bar{\to}_R)$ eine Reduktionsbasis von $(\Delta, \equiv, \to_R)$ ist. Dazu nehmen wir $E \in \Delta^\alpha$ und $E \to_R F$ an:

$E \to_R F$	dann	$E\ (\equiv \circ \bar{\to}_R \circ \equiv)\ F$	Proposition 5.1.1
	dann	$E\ (\equiv_1 \circ {}^*_\alpha\overset{s}{\Leftarrow} \circ \bar{\to}_R \circ \equiv)\ F$	Proposition 5.2.7
	dann	$E\ (\equiv_1 \circ \bar{\to}_R \circ {}^*_\alpha\Leftarrow \circ \equiv)\ F$	Proposition 5.2.8
	dann	$E\ (\equiv_1 \circ \bar{\to}_R \circ \equiv)\ F$	

□

5.2.4 Beweise der Details bezüglich $\overset{s}{\Rightarrow}_\alpha$

Wir führen nun die Beweise der Proposition 5.2.6 und 5.2.8, müssen also etliche Vertauschungseigenschaften für $\overset{s}{\Rightarrow}_\alpha$ zeigen. Dazu passen wir die Theorie der Termersetzungsysteme [DJ90] für unsere Zwecke an.

Lagebeziehungen von Teilausdrücken.

Seien R_1 und R_2 binäre Relationen auf Δ. Dann betrachten wir die folgende Situation:

$$F_1 \ {}_{R_1}\!\Leftarrow E \Rightarrow_{R_2} F_2 \,.$$

Wir sagen, daß R_1 oberhalb von R_2 angewendet wird, falls E_1, E_2, E_3, E_1' und E_2' mit folgenden Eigenschaften existieren:

$$\begin{array}{lcl} E & = & E_1[E_2[\![E_3]\!]]\,, \\ E_2' & {}_{R_1}\!\leftharpoondown & E_2 \quad \text{und} \quad E_3 \ \rightharpoondown_{R_2} \ E_3'\,, \\ F_1 & = & E_1[E_2'[\![E_3]\!]]\,, \\ F_2 & = & E_1[E_2[\![E_3']\!]]\,. \end{array}$$

Die Situation von Anwendung von R_1 oberhalb R_2 veranschaulichen wir in Abbildung 5.2.

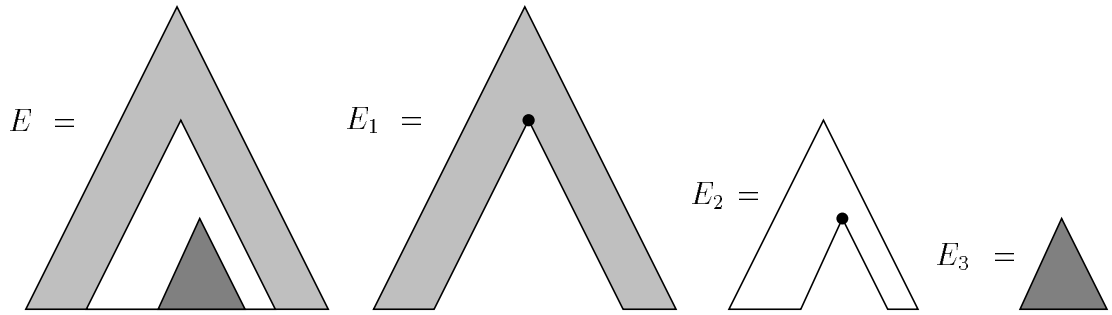


Abbildung 5.2: $E = E_1[E_2[\![E_3]\!]]$

Die Relation R_1 wird unterhalb von R_2 angewendet, falls R_2 oberhalb von R_1 angewendet wird. Die Relationen R_1 und R_2 werden in gleicher Position angewendet, falls E_1, E_2, E_1', E_2' und E_2'' mit den folgenden Eigenschaften existieren:

$$\begin{array}{lcl} E & = & E_1[E_2]\,, \\ E_2' & {}_{R_1}\!\leftharpoondown & E_2 \ \rightharpoondown_{R_2} \ E_2''\,, \\ F_1 & = & E_1[E_2']\,, \\ F_2 & = & E_1[E_2'']\,. \end{array}$$

Die Relationen R_1 und R_2 werden in nebeneinander angewendet, wenn E_1, E_2, E_3, E_2' und E_3' mit folgenden Eigenschaften existieren:

$$\begin{array}{lcl} E & = & E_1[E_2 \wedge E_3]\,, \\ E_2' & {}_{R_1}\!\!\leftharpoondown & E_2 \quad \text{und} \quad E_3 \rightharpoondown_{R_2} E_3'\,, \\ F_1 & = & E_1[E_2' \wedge E_3]\,, \\ F_2 & = & E_1[E_2 \wedge E_3']\,. \end{array}$$

Nebeneinanderliegendende Positionen werden in Abbildung 5.3 veranschaulicht[2].

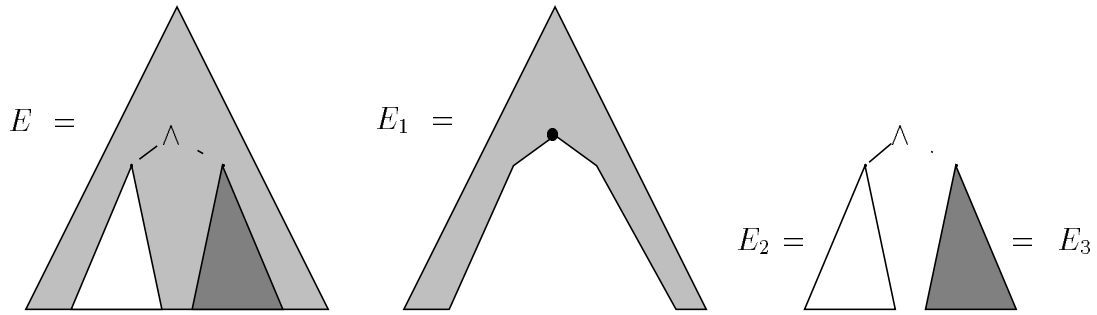


Abbildung 5.3: $E = E_1[\![E_2 \wedge E_3]\!]$

Lemma 5.2.9 *In der Situation* $F_1 \;{}_{R_1}\!\!\Leftarrow E \Rightarrow_{R_2} F_2$ *wird* R_1 *oberhalb oder unterhalb von* R_2 *angewendet oder aber* R_1 *und* R_2 *werden in gleicher Position oder nebeneneinander angewendet.*

Beweis. Mit struktureller Induktion über E unter Verwendung von Proposition 5.2.4. □

Beweise der Vertauschungseigenschaften von $\overset{s}{\Rightarrow}_\alpha$.

Wir weisen nun Vertauschungseigenschaften von $\overset{s}{\Rightarrow}_\alpha$ nach, indem wir jeweils alle möglichen Lagebeziehungen untersuchen. Dabei müssen wir insbesondere mit Substitutionsoperatoren $[x/y]$ oder $[\overline{x}/\overline{y}]$ und Nebenbedingungen der Art $x \notin \mathcal{BV}(E)$

[2]In unserer Definition von nebeneinanderliegenden Positionen nützen wir aus, daß Ausdrücke Terme sind, die nur einen einzigen mehrstelligen Konstruktor enthalten, nämlich den Kompositionskonstruktor $\wedge$. Dennoch läßt sich unsere Definitionsmethode auch für Termersetzungssysteme mit beliebiger Signatur anwenden. In diesem Fall liegen zwei Positionen nebeneinander, wenn ein mehrstelliger Konstruktor f in der Signatur existiert, der sich analog zu $\wedge$ in der obigen Definition verhält. Dies zeigt, daß sich Kontexte als interessante Alternative zu Pfaden [DJ90] anbieten. Die Idee zur Definition von nebeneinanderliegenden Positionen mithilfe von Kontexten verdanke ich Martin Müller.

zurecht kommen. Dazu verwenden wir den Begriff der Substitutionsinvarianz aus Definition 3.5.14 in Kombination mit dem Begriff der Sicherheit, den wir nun definieren.

Definition 5.2.10 *Eine binäre Relation R auf Ausdrücken heißt* sicher, *wenn sie von links nach rechts angewendet keine neuen freien oder gebundenen Variablen erzeugt, also wenn mit $(E, F) \in R$ auch $\mathcal{BV}(F) \subseteq \mathcal{BV}(E)$ und $\mathcal{FV}(F) \subseteq \mathcal{FV}(E)$ gelten.*

Proposition 5.2.11 *Sei R sicher und gelte $F_1 \ {}_R\Leftarrow E \stackrel{s}{\Rightarrow}_\alpha F_2$, wobei R und (α) nebeneinander angewendet werden. Dann existiert G mit $F_1 \stackrel{s}{\Rightarrow}_\alpha G \ {}_R\Leftarrow E_2$. Das gleiche Resultat gilt auch für $\bar{\to}_R$ anstelle von $\Rightarrow_R$.*

Beweis. Per Definition existieren E_1, E_2, E_3, E'_2 und E'_3 mit folgenden Eigenschaften:

$$\begin{array}{rcl} E & = & E_1[\![E_2 \wedge E_3]\!]\,, \\ E'_2 & {}_R\bar{\leftarrow} & E_2 \quad \text{und} \quad E_3 \ \bar{\rightharpoonup}_\alpha \ E'_3\,, \\ F_1 & = & E_1[\![E'_2 \wedge E_3]\!]\,, \\ F_2 & = & E_1[\![E_2 \wedge E'_3]\!]\,. \end{array}$$

Setzen wir $G = E_1[\![E'_2 \wedge E'_3]\!]$, dann gilt $F_1 \stackrel{s}{\Rightarrow}_\alpha G$ wegen der Sicherheit von R und $G \ {}_R\Leftarrow F_2$ trivialerweise. □

Proposition 5.2.12 *Sei R sicher und invariant unter Substitution und gelte $F_1 \ {}_R\Leftarrow E \stackrel{s}{\Rightarrow}_\alpha F_2$, wobei (α) oberhalb von R anwendet wird. Dann existiert G mit $F_1 \stackrel{s}{\Rightarrow}_\alpha G \ {}_R\Leftarrow E_2$. Das gleiche Resultat gilt auch für $\bar{\to}_R$ anstelle von $\Rightarrow_R$.*

Beweis. Wir führen nur den Fall für α-Umbenennung von deklarierten Varialben mittels $\stackrel{s}{\Rightarrow}_\alpha$ aus. Nach Definition existieren E_1, E_2, E_3, E'_2, E'_3 x, y mit folgenden Eigenschaften:

$$\begin{array}{rcl} E & = & E_1[\![\exists x\, E_2[\![E_3]\!]]\!] \\ E_3 \ {}_R\bar{\leftarrow} & E'_3 \quad \text{und} \quad \exists x\, E_2 \ \bar{\rightharpoonup}_\alpha & \exists y\, E'_2[y/x] \\ F_1 & = & E_1[\![\exists x\, E_2[\![E'_3]\!]]\!] \\ F_2 & = & E_1[\![(\exists y\, E'_2[y/x])[\![E_3]\!]]\!] \\ y \notin \mathcal{V}(E) & & \end{array}$$

Aus der Sicherheit von R folgt $\mathcal{V}(E'_3) \subseteq \mathcal{V}(E_3)$, also $y \notin \mathcal{V}(F_1)$ und somit auch:

$$F_1 \ \stackrel{s}{\Rightarrow}_\alpha \ E_1[\exists y(E_2[\![E'_3]\!])[y/x]] \ = \ G\,.$$

Dadurch haben wir einen Kandidaten G gefunden. Aus der Invarianz unter Substitution von R folgt $E_3[y/x] \bar{\rightarrow}_R E_3'[y/x]$. Nun können wir die Behauptung folgendermaßen schließen:

$$\begin{array}{rcl} F_2 & = & E_1[\![(\exists y\, E_2'[y/x])[\![E_3]\!]]\!] \\ & = & E_1[\![(\exists y\, E_2'[y/x])[\![E_3[y/x]]\!]]\!] \\ & \Rightarrow_R & E_1[\![(\exists y\, E_2'[y/x])[\![E_3'[y/x]]\!]]\!] \\ & = & E_1[\![\exists y (E_2'[\![E_3']\!])[y/x]]\!] \\ & = & G \end{array}$$

Alle anderen Fälle lassen sich ähnlich nachweisen oder treten gar nicht auf, da $\bar{\rightarrow}_R$ nur in schwachen Kontexten angewendet werden darf. □

Wir erinnern daran, daß wir Axiome auch frei als binäre Relationen auffassen indem wir das Relationszeichen des Axioms ignorieren. Dadurch übertragen sich die Begriffe Sicherheit und Substitutionsinvarianz auf Axiome.

Proposition 5.2.13 *1. Alle Axiomen der strukturellen Kongruenz und Reduktion des δ-Kalküls inklusive Annullierung sind invariant unter Substitution.*

2. Sei R invariant unter Substitution. Dann sind auch $\bar{\rightarrow}_R$, R^{-1}, $\Rightarrow_R$ und $\equiv_R$ invariant unter Substitution.

3. Die Relationen $\bar{\rightarrow}_A$, $\bar{\rightarrow}_E$ und $\bar{\rightarrow}_G$ sind sicher, aber deren Inverse nicht. Die Axiome (ACI), (Exch) und (Scope) sind sicher, genauso wie ihre Inversen[3]. Das Axiom (α) ist nicht sicher.

Beweis.

1. Die Nachweise sind aufwendig und laufen auf Substitutionsgleichungen hinaus, die wir bereits in Abschnitt 3.5.1 nachgewiesen haben.

   (a) **Fall** (A): Sei $E \bar{\rightarrow}_A F$ und $[w'/w]$ eine Substitution mit $w' \notin \mathcal{BV}(E)$. Dann haben E und F die folgenden Formen:

$$\begin{array}{rcl} E & = & x{:}\overline{y}/G \wedge x\,\overline{u}\ , \\ F & = & x{:}\overline{y}/G \wedge G[\overline{u}/\overline{y}]\ . \end{array}$$

[3]Das Axioms $\exists x \top \equiv \top$ ist nicht sicher. Dies ist der Grund weshalb wir dieses Axiom im Gegensatz zu neueren Darstellungen des π-Kalküls [Mil91] nicht zur strukturellen Kongruenz hinzunehmen

Aus der Nebenbedingung von (A) folgt $\mathcal{V}(\overline{u}) \cap \mathcal{BV}(G) = \emptyset$. Die Behauptung erhalten wir wie folgt:

$$\begin{array}{rcl} E[w'/w] & = & x[w'/w]{:}\overline{y}[w'/w]/G[w'/w] \wedge x[w'/w]\,\overline{u}[w'/w] \\ & \dot{\rightarrow}_A & (x{:}\overline{y}/G)[w'/w] \wedge G[w'/w][\overline{u}[w'/w]/\overline{y}[w'/w]] \\ & = & (x{:}\overline{y}/G)[w'/w] \wedge G[\overline{u}/\overline{y}][w'/w] \\ & = & F[w'/w] \end{array}$$

Die Nebenbedingung von (A) vererbt sich, da w' in E nicht gebunden vorkommt. Desweiteren haben wir die folgende Substitutionsgleichung verwendet, die durch Lemma 3.5.11 gerechtfertigt wird:

$$[w'/w] \circ [\overline{u}[w'/w]/\overline{y}[w'/w]] \;=\; [\overline{u}/\overline{y}] \circ [w'/w]\,.$$

Die Voraussetzung dieses Lemmas $w' \notin \mathcal{V}(\overline{y})$ folgt aus $w' \notin \mathcal{BV}(E)$.

(b) **Fall** (E)**:** Wir setzen nun $E \dot{\rightarrow}_E F$ und eine Substitution $[w'/w]$ mit $w' \notin \mathcal{B}(E)$ voraus. Dann haben E und F die folgenden Formen:

$$\begin{array}{rcl} E & = & \exists x \exists y (x = y \wedge G)\,, \\ F & = & \exists y\, G[y/x]\,. \end{array}$$

Aus den Nebenbedingungen von (E) folgt $x \neq y$ und $x \notin \mathcal{B}(G)$. Wir können nun wie folgt schließen:

$$\begin{array}{rcl} E[w'/w] & = & \exists x \exists y (x[w'/w] = y[w'/w] \wedge G[w'/w]) \\ & \dot{\rightarrow}_E & \exists y (G[w'/w][y[w'/w]/x[w'/w]]) \\ & = & \exists y (G[y/x][w'/w]) \\ & = & F[w'/w] \end{array}$$

Die Nebenbedingungen vereben sich erneut wegen $w' \notin \mathcal{BV}(E)$. Die Substitutionsgleichung

$$[w'/w][y[w'/w]/x[w'/w]] \;=\; [y/x][w'/w]$$

wird wiederum durch Lemma 3.5.11 gerechtfertigt.

(c) **Fall** (α)**:** Wir betrachten wiederum nur die Umbennung von deklaratierten Variablen. Dazu nehmen wir $E \dot{\rightarrow}_\alpha F$ und $w' \notin \mathcal{BV}(E) \cup \mathcal{BV}(F)$ an, wobei E und F die folgenden Formen haben:

$$\begin{array}{rcl} E & = & \exists x\, G\,, \\ F & = & \exists y (G[y/x])\,. \end{array}$$

Als der Nebenbedingung von (α) erhalten wir wir $y \notin \mathcal{V}(G)$.

$$\begin{array}{rcl} E[w'/w] & = & \exists x[w'/w]G[w'/w] \\ & \bar{\rightarrow}_\alpha & \exists y[w'/w]G[w'/w][y[w'/w]/x[w'/w]] \\ & = & \exists y[w'/w]G[y/x][w'/w] \\ & = & (\exists y\, G[y/x])[w'/w] \\ & = & F[w'/w] \end{array}$$

Die Nebenbedingung für (α) vererbt sich wegen $w' \notin \mathcal{BV}(E) \cup \mathcal{BV}(F)$. Die Substitutionsgleichung

$$[w'/w][y[w'/w]/x[w'/w]] \;=\; [y/x][w'/w]$$

wird wiederum durch Lemma 3.5.11 gerechtfertigt.

(d) Alle anderen Fälle sind analog oder einfacher, da sie keine Substitutionen benutzen.

2. Dieser Teil ist einfach. Die Relation R^{-1} ist invariant unter Substitution, weil die Definition von Substitutionsinvarianz symmetrisch ist.

3. Die positiven Aussagen sind einfach. Nur im Falle von (*Scope*) müssen wir auf Nebenbedingungen achten. Ein Gegenbeispiel zur Sicherheit von ${}_A\bar{\leftarrow}$ liefert:

$$x{:}y/\top \;\; {}_A\bar{\leftarrow} \;\; x{:}y/\top \wedge x\, y \;.$$

□

Beweis von Proposition 5.2.8. Sei R eine der Relationen (A), (E) oder (G). Dann müssen wir die folgende Eigenschaft zeigen:

$$({}^*_\alpha\overset{s}{\Leftarrow} \circ \bar{\rightarrow}_R) \;\subseteq\; (\bar{\rightarrow}_R \circ {}^*_\alpha\Leftarrow)\,.$$

Diese ist äquivalent zu der nachstehenden, wie sich mit Induktion über die Anzahl der Schritte in ${}^*_\alpha\overset{s}{\Leftarrow}$ nachweisen läßt:

$$({}_\alpha\overset{s}{\Leftarrow} \circ \bar{\rightarrow}_R) \;\subseteq\; (\bar{\rightarrow}_R \circ {}^*_\alpha\Leftarrow)\,.$$

Zum Beweis dieser Eigenschaft betrachten wir die Situtation:

$$F_1 \;\; {}_\alpha\overset{s}{\Leftarrow} \;\; E \;\; \bar{\rightarrow}_R \;\; F_2\,.$$

Nach Lemma 5.2.9 gibt es genau vier mögliche Lagebeziehungen für die Anwendungen von R und (α). Werden R und (α) nebeneinander angewendet, dann existiert nach Proposition 5.2.11 und Proposition 5.2.13 ein G mit:

$$F_1 \;\; \bar{\rightarrow}_R \;\; G \;\; {}_\alpha\overset{s}{\Leftarrow} \;\; F_2\,.$$

Gleiches gilt wegen der Proposition 5.2.12 und 5.2.13, falls (α) oberhalb von R angewendet wird.

Nun betrachten wir den Fall, daß R und (α) in der gleichen Position angewendet werden. Im Fall $R = (A)$ ist dies ausgeschlossen, da der oberste Konstruktor eines Ausdruckes, auf den (A) anwendbar ist, der Kompositionskonstruktor $\wedge$ ist. Im Fall $R = (E)$ existieren x, y, z, E_1 und E_2 mit:

$$\begin{array}{rcl} E & = & E_1[\![\exists x \exists y (x = y \wedge E_2)]\!] \,, \\ F_1 & = & E[\![\exists z ((\exists y (x = y \wedge E_2))[z/x])]\!] \,, \\ F_2 & = & E[\![\exists y E_2[y/x]]\!] \,. \end{array}$$

Dabei gilt $z \notin \mathcal{V}(E)$, $x \neq y$ und $y \notin \mathcal{BV}(E_2)$. Die Behauptung $F_1(\bar{\to}_E \circ {}^{*}_{\alpha}\Leftarrow)F_2$ folgt aus:

$$\begin{array}{rcll} F_1 & = & E_1[\![\exists z ((\exists y (x = y \wedge E_2))[z/x])]\!] & \\ & = & E_1[\![\exists z \exists y (z = y \wedge E_2[z/x])]\!] & \text{da } y \neq x \\ & \bar{\to}_E & E_1[\![\exists y (E_2[z/x][y/z])]\!] & \text{da } z \neq x \text{ und } z \notin \mathcal{BV}(E_2) \\ & = & E_1[\![\exists y (E_2[y/x])]\!] & \text{da } z \notin \mathcal{FV}(E_2) \\ & = & F_2 & \end{array}$$

Gleiche Positionen von Annullierung (G) und (α) sind genauso einfach.

Wir betrachten nun den Fall, daß (α) unterhalb von R angewendet wird. Wiederum beginnen wir mit Applikation, also mit $R = (A)$.

1. **Fall** (α) wird im Rumpf der applizierten Abstraktion angewendet: Dann existieren E_1, E_2 E_2', x und $\overline{y}$ mit folgenden Eigenschaften:

$$\begin{array}{rcl} E & = & E_1[\![x{:}\overline{y}/E_2 \wedge x\,\overline{u}\,]\!] \\ F_1 & = & E_1[\![x{:}\overline{y}/E_2' \wedge x\,\overline{u}\,]\!] \\ F_2 & = & E_1[\![x{:}\overline{y}/E_2 \wedge E_2[\overline{u}/\overline{y}]]\!] \\ E_2 & \overset{s}{\Rightarrow}_{\alpha} & E_2' \end{array}$$

   Ist w diejenige Variable, die bei Umbenennung in E_2 eingeführt wird, dann gilt $w \notin \mathcal{V}(E)$. Aus $\mathcal{V}(\overline{u}) \cap \mathcal{BV}(E_2) = \emptyset$ folgt wegen $\mathcal{BV}(E_2') \subseteq \mathcal{BV}(E_2) \cup \{w'\}$ auch $\mathcal{V}(\overline{u}) \cap \mathcal{BV}(E_2') = \emptyset$, so daß wir wie folgt applizieren können:

$$F_1 \quad \bar{\to}_A \quad E_1[\![x{:}\overline{y}/E_2' \wedge E_2'[\overline{u}/\overline{y}]]\!] \;=\; G.$$

   Wegen Substitionsinvarianz von (α) (Proposition 5.2.13) folgt $E_2[\overline{u}/\overline{y}] \Rightarrow_{\alpha} E_2'[\overline{u}/\overline{y}]$. Nun können wir wie folgt schließen:

$$\begin{array}{rcl} F_2 & \Rightarrow_{\alpha} & E_1[\![x{:}\overline{y}/E_2' \wedge E_2[\overline{u}/\overline{y}]]\!] \\ & \Rightarrow_{\alpha} & E_1[\![x{:}\overline{y}/E_2' \wedge E_2'[\overline{u}/\overline{y}]]\!] \\ & = & G \end{array}$$

2. **Fall** (α) wird auf ein formales Argumenten der beteiligten Abstraktion angewendet: Dann haben wir folgende Situation vorliegen:

$$\begin{array}{rcl} E & = & E_1[\![x{:}\overline{v}y\overline{w}/E_2 \wedge x\,\overline{u'}y'\overline{w'}]\!]\,, \\ F_1 & = & E_1[\![x{:}\overline{v}z\overline{w}[z/y]/E_2[z/y] \wedge x\,\overline{u'}y'\overline{w'}]\!]\,, \\ F_2 & = & E_1[\![x{:}\overline{v}y\overline{w}/E_2 \wedge E_2[\overline{u'}y'\overline{v'}/\overline{u}y\overline{v}]]\!]\,. \end{array}$$

Desweiteren gilt $z \notin \mathcal{V}(E)$ und $\overline{v}y\overline{w}$ ist linear. Somit ist auch $\overline{v}z\overline{w}$ linear und es gilt $\overline{w}[z/y] = \overline{w}$. Nun folgt:

$$\begin{array}{rcl} F_1 & \bar{\to}_A & E_1[\![x{:}\overline{v}z\overline{w}/E_2[z/y] \wedge E_2[z/y][\overline{u'}y'\overline{w'}/\overline{v}z\overline{w}]] \\ & = & E_1[\![x{:}\overline{v}z\overline{w}/E_2[z/y] \wedge E_2[\overline{u'}y'\overline{w'}/\overline{v}y\overline{w}]]\!] \\ F_2 & \Rightarrow_\alpha & E_1[\![x{:}\overline{v}z\overline{w}[z/y]/E_2[z/y] \wedge E_2[\overline{u'}y'\overline{v'}/\overline{u}y\overline{v}]]\!] \end{array}$$

Die Betrachtungen für (α) unterhalb von (E) oder (G) führen wir hier nicht aus. Sie verlaufen ähnlich zu denjenigen für (A). □

Als letze Vorbereitung zum Beweis von Proposition 5.2.6 notieren wir das folgende Lemma:

Lemma 5.2.14 $\overset{s}{\Rightarrow}_\alpha$ *ist konfluent.*

Beweis. Mit struktureller Induktion über Ausdrücke. Es ist bemerkenswert, daß $\overset{s}{\Rightarrow}_\alpha$ nicht stark konfluent ist, wie man an folgenden Beispiel sieht, in dem wir x und u als verschieden voraussetzen:

$$\exists x \exists u\, u = u \quad {}_\alpha\overset{s}{\Leftarrow} \quad \exists x \exists x\, x = x \quad \overset{s}{\Rightarrow}_\alpha \quad \exists u \exists x\, x = x\,.$$

□

Beweis von Proposition 5.2.6.

1. Es gilt $\equiv_\alpha \subseteq (\overset{s}{\Rightarrow}_\alpha^* \circ {}_\alpha^*\overset{s}{\Leftarrow})$:

   Die Relation $\equiv_\alpha$ ist die kleinste Relation die folgende Regeln erfüllt:

$$\frac{E \Rightarrow_\alpha F}{E \equiv_\alpha F} \qquad \frac{E \;{}_\alpha\!\Leftarrow F}{E \equiv_\alpha F} \qquad \frac{E \equiv_\alpha E' \qquad E' \equiv_\alpha F}{E \equiv_\alpha F}$$

   Wir zeigen die Behauptung mit Induktion über Herleitungen von $E \equiv_\alpha F$ bezüglich dieser Regeln. Wir nehmen also $E \equiv_\alpha F$ an. Im Falle der ersten Regel liegt dann die folgende Situation vor:

$$E \;=\; E_1[\![\exists x\, E_2]\!] \;\Rightarrow_\alpha\; E_1[\![\exists y\, E_2[y/x]]\!] \;=\; F$$

und $y \notin \mathcal{V}(E_2)$. Ist u eine frische Variable dann gilt:

$$E \ \overset{s}{\Rightarrow}_\alpha \ E_1[\![\exists u\, E_2[u/x]]\!] \ = \ E_1[\![\exists u\, E_2[y/x][u/y]]\!] \ {}_\alpha\overset{s}{\Leftarrow} \ F \, .$$

Somit haben wir $E \ (\overset{s}{\Rightarrow}_\alpha \circ {}_\alpha\overset{s}{\Leftarrow}) \ F$ gezeigt. Der Beweis für die zweite Regel ist symmetrisch. Für die dritte Regel nehmen wir $E \equiv_\alpha E'$ und $E' \equiv_\alpha F$ an. Aus der Induktionsvoraussetzung erhalten wir:

$$E \ (\overset{s}{\Rightarrow}{}^*_\alpha \circ {}^*_\alpha\overset{s}{\Leftarrow} \circ \overset{s}{\Rightarrow}{}^*_\alpha \circ {}^*_\alpha\overset{s}{\Leftarrow}) \ F \, .$$

Nach Lemma 5.2.14 ist $\overset{s}{\Rightarrow}_\alpha$ konfluent, woraus $E(\overset{s}{\Rightarrow}{}^*_\alpha \circ {}^*_\alpha\overset{s}{\Leftarrow}) \ F$ folgt.

2. Aus $E \overset{s}{\Rightarrow}_\alpha F$ und $E \in \Delta^\alpha$ folgt $E \ {}_\alpha\overset{s}{\Leftarrow} F$:

   Dies läßt sich unmittelbar aus der folgenden Eigenschaft ableiten, da zwei Ausdrücke E und F mit $E \overset{s}{\Rightarrow}_\alpha F$ verschieden sein müssen.

   Wenn $E \in \Delta^\alpha$ und $E \Rightarrow_\alpha F$ dann gilt $E \ {}_\alpha\overset{s}{\Leftarrow} F$ oder $E = F$.

   Dies läßt sich mit Induktion über Herleitungen von $E \Rightarrow_\alpha F$ nachweisen. Sei also $E \in \Delta^\alpha$ mit $E \Rightarrow_\alpha F$. Dann liegt die folgende Situation vor:

$$\begin{array}{rcl} E & = & E_1[\![E_2]\!] \, , \\ F & = & E_1[\![E_2']\!] \, , \\ E_2 & \overset{-}{\rightharpoonup}_\alpha & E_2' \, . \end{array}$$

   Wir betrachten nun lediglich den Fall, daß eine deklarierte Variabele umbenannt wird. Dann gilt

$$\begin{array}{rcl} E_2 & = & \exists x\, E_3 \, , \\ E_2' & = & \exists y\, E_3[y/x] \, . \end{array}$$

   Außerdem wissen wir $y \notin \mathcal{V}(E_3)$. Wir können $y \neq x$ annehmen, da anderenfalls $E = F$ gilt. Aus $E \in \Delta^\alpha$ folgt $x \notin \mathcal{BV}(E_3)$. Daraus und wegen $x \neq y$ folgt $x \notin \mathcal{V}(E_3[y/x])$ und somit gilt auch:

$$E_2' \ = \ \exists y\, E_3[y/x] \ \overset{-}{\rightharpoonup}_\alpha \ \exists x\, E_3[y/x][x/y] \ = \ \exists x\, E_3 \ = \ E_2 \, .$$

   Aus $E \in \Delta^\alpha$ erhalten wir $x \notin \mathcal{V}(E_1)$ und somit folgt $x \notin \mathcal{V}(E_1[\![E_2']\!])$, d.h. $x \notin \mathcal{V}(F)$. Deshalb gilt sogar:

$$F \ = \ E_1[\![E_2']\!] \ \overset{s}{\Rightarrow}_\alpha \ E_1[\![E_2]\!] \ = \ E \, .$$

3. In Analogie zu Lemma 5.2.3 ist einfach zu zeigen, daß $\overset{s}{\Rightarrow}_\alpha$ α-Standardisierung erhält. Der Beweis von $({}^*_\alpha\overset{s}{\Leftarrow} \circ \equiv_1) \subseteq (\equiv_1 \circ {}^*_\alpha\Leftarrow)$ ist ähnlich zu demjenigen von Proposition 5.2.8. □

5.3 Basen durch Pränexnormalformen

In diesem Abschnitt beweisen wir die Korrektheit von der Sätze 3.4.3 und 3.4.4 aus Abschnitt 3.4. Diese geben Definitions- und Reduktionsbasen des δ-Kalküls an, die auf α-standardisierten Pränexnormalformen (PNFs) aufgebaut sind.

5.3.1 Definitionsbasen

Abbildung 5.4 wiederholt die Definition von Pränexnormalformen (PNFs) aus Kapitel 3.4. Desweiteren erinnern wir daran, daß eine Normalform eine α-standardisierte PNF ist und daß wir die Menge aller Normalformen mit Δ^{nf} denotieren.

B	$::=$	$x{:}\overline{y}/D \mid x\,\overline{y} \mid x = y$	Moleküle
C	$::=$	$\top \mid B \mid C \wedge C$	Chemische Lösungen
D	$::=$	$C \mid \exists x\, D$	PNFs

Abbildung 5.4: PNFs

In Abbildung 5.5 definieren wir eine Funktion $[\,\_\,] : \Delta^{\alpha} \to \Delta^{nf}$, die α-standardisierte Ausdrücke auf ihre Normalform abbildet. Diese Normalform wird berechnet, indem alle Deklarationen so weit wie möglich nach außen geschoben werden. Da wir α-standardisierte Ausdrücke voraussetzen, ist Kapern von Variablen ausgeschlossen.

$[\top] = \top$	$[x = y] = x = y$	$[x{:}\overline{y}/E] = x{:}\overline{y}/[E]$
$[x\,\overline{y}\,] = x\,\overline{y}$	$[\exists x\, E] = \exists x [E]$	$\dfrac{[E_1] = \exists\overline{x_1} C_1 \quad [E_2] = \exists\overline{x_2} C_2}{[E_1 \wedge E_2] = \exists\overline{x_1}\exists\overline{x_2}(C_1 \wedge C_2)}$

Abbildung 5.5: Berechnung von Normalformen

Offensichtlich existiert zu jedem E ein eindeutig bestimmtes F mit $[E] = F$ und somit ist $[\,\_\,]$ eine wohldefinierte Funktion.

Lemma 5.3.1 *Für alle E ist $[E]$ eine PNF.*

Beweis. Mit Induktion über Herleitungen von $[E] = F$. □

Lemma 5.3.2 *Für alle PNFs E gilt $[E] = E$.*

Beweis. Mit Induktion über die Struktur von PNFs. □

Lemma 5.3.3 *$\equiv_2$ und $\stackrel{s}{\Rightarrow}_\alpha$ erhalten PNFs und somit auch Normalformen.*

Beweis. Mit Induktion über Herleitungen von $E \equiv_2 F$ beziehungsweise $E \stackrel{s}{\Rightarrow}_\alpha F$. □

Lemma 5.3.4 *Für alle $E \in \Delta^\alpha$ gilt $E \equiv_1 [E]$.*

Beweis. Mit struktureller Induktion über E zeigen wir, daß für alle F mit $[E] = F$ auch $E \equiv_1 F$ gilt. Der Beweis ist einfach und deshalb spielen wir nur den Fall einer Komposition durch. Dazu sei $E = E_1 \wedge E_2$ und es gelte $E \in \Delta^\alpha$. Wir können nur eine einzige Regel anwenden, um $[E] = F$ herzuleiten:

$$\frac{[E_1] = \exists \overline{x_1} C_1 \qquad [E_2] = \exists \overline{x_2} C_2}{[E_1 \wedge E_2] = \exists \overline{x_1} \exists \overline{x_2} (C_1 \wedge C_2)}$$

Dann gilt $F = \exists \overline{x_1} \exists \overline{x_2} (C_1 \wedge C_2)$. Aus der Induktionsannahme angewendet auf E_1 und E_2 erhalten wir 1) $E_1 \equiv_1 \exists \overline{x_1} C_1$ und 2) $E_2 \equiv_1 \exists \overline{x_2} C_2$. Wir schließen die Behauptung wie folgt:

$$\begin{array}{lll}
E_1 \wedge E_2 & \equiv_1 & (\exists \overline{x_1} C_1) \wedge (\exists \overline{x_2} C_2) \qquad \text{wegen 1) und 2)} \\
& \equiv_{Scope} & \exists \overline{x_1} (C_1 \wedge \exists \overline{x_2} C_2) \qquad \text{da } \mathcal{V}(\overline{x_1}) \cap \mathcal{FV}(\exists \overline{x_2} C_2) = \emptyset \\
& \equiv_{ACI} & \exists \overline{x_1} ((\exists \overline{x_2} C_2) \wedge C_1) \\
& \equiv_{Scope} & \exists \overline{x_1} \exists \overline{x_2} (C_2 \wedge C_1) \qquad \text{da } \mathcal{V}(\overline{x_2}) \cap \mathcal{FV}(C_1) = \emptyset \\
& \equiv_{ACI} & \exists \overline{x_1} \exists \overline{x_2} (C_1 \wedge C_2) \\
& = & F
\end{array}$$

Dies beweist $E_1 \wedge E_2 \equiv_1 F$ unter der Annahme, daß die Randbedingungen für die obigen Anwendung von (*Scope*) erfüllt sind. Wir müssen also $\mathcal{V}(\overline{x_1}) \cap \mathcal{FV}(\exists \overline{x_2} C_2) = \emptyset$ und $\mathcal{V}(\overline{x_2}) \cap \mathcal{FV}(C_1) = \emptyset$ überprüfen. Die Formeln 1) und 2) kombiniert mit Lemma 5.2.2 implizieren:

$$\begin{array}{ll}
\mathcal{FV}(E_1) = \mathcal{FV}(\exists \overline{x_1} C_1), & \mathcal{BV}(E_1) = \mathcal{BV}(\exists \overline{x_1} C_1), \\
\mathcal{FV}(E_2) = \mathcal{FV}(\exists \overline{x_2} C_2), & \mathcal{BV}(E_2) = \mathcal{BV}(\exists \overline{x_2} C_2).
\end{array}$$

Wegen α-Standardisierung gilt $\mathcal{FV}(E_1 \wedge E_2) \cap \mathcal{BV}(E_1 \wedge E_2) = \emptyset$. Die benötigen Bedingungen erhalten wir wie folgt:

$$\begin{array}{l}
\mathcal{FV}(\exists \overline{x_2} C_2) \;=\; \mathcal{FV}(E_2) \;\subseteq\; \mathcal{FV}(E_1 \wedge E_2)\,, \\
\mathcal{V}(\overline{x_1}) \;\subseteq\; \mathcal{BV}(\exists \overline{x_1} C_1) \;=\; \mathcal{BV}(E_1) \;\subseteq\; \mathcal{BV}(E_1 \wedge E_2)\,, \\
\mathcal{BV}(\overline{x_2}) \;\subseteq\; \mathcal{BV}(\exists \overline{x_2} C_2) \;=\; \mathcal{BV}(E_2) \;\subseteq\; \mathcal{BV}(E_1 \wedge E_2)\,, \\
\mathcal{FV}(C_1) \;\subseteq\; \mathcal{FV}(\exists \overline{x_1} C_1) \;=\; \mathcal{FV}(E_1) \;\subseteq\; \mathcal{FV}(E_1 \wedge E_2)\,.
\end{array}$$

□

Proposition 5.3.5 (Skopus) *Aus $E \equiv_1 F$ folgt $[E] \equiv_2 [F]$.*

Beweis. Wir müssen $[\tilde{E}] \equiv_2 [\tilde{G}]$ unter der Annahme $\tilde{E} \equiv_1 \tilde{G}$ nachweisen. Dies tun wir mit Induktion über Herleitungen von $\tilde{E} \equiv_1 \tilde{G}$, wobei wir wiederum nur die interessantesten Fälle durchspielen.

Wir beginnen mit einer der beiden Regeln für Komposition. Dazu sei $\tilde{E} = E_1 \wedge G$ und $\tilde{F} = F \wedge G$.

$$\frac{E_1 \equiv_1 F}{E_1 \wedge G \equiv_1 F \wedge G}$$

Wir müssen nun $[E_1 \wedge G] \equiv_2 [F \wedge G]$ zeigen. Aus der Induktionsannahme folgt $[E_1] \equiv_2 [F]$. Wegen Lemma 5.3.1 gibt es chemische Lösungen C_1, C_2 und C mit

$$[E_1] = \exists\overline{x_1}C_1 \qquad [F] = \exists\overline{x_2}C_2 \qquad [G] = \exists\overline{x}C$$

Durch zweifach Anwendung von Proposition 3.5.1 und mit Lemma 3.5.7 erhalten wir die Äquivalenz der folgenden Zeilen:

$$\begin{array}{ll} & [E_1 \wedge G] \equiv_2 [F \wedge G] \\ \text{gdw.} & \exists\overline{x_1}\exists\overline{x}(C_1 \wedge C) \equiv_2 \exists\overline{x_2}\exists\overline{x}(C_2 \wedge C) \\ \text{gdw.} & \exists\overline{x_1}\exists\overline{x} \approx \exists\overline{x_2}\exists\overline{x} \text{ und } C_1 \wedge C \approx C_2 \wedge C \\ \text{gdw.} & \exists\overline{x_1} \approx \exists\overline{x_2} \text{ und } C_1 \approx C_2 \\ \text{gdw.} & \exists\overline{x_1}C_1 \equiv_2 \exists\overline{x_2}C_2 \\ \text{gdw.} & [E] \equiv_2 [F] \end{array}$$

Die letzte Zeile folgt aus der Induktionsvoraussetzung und somit haben wir die erste gezeigt. □

Beweis von Satz 3.4.3. Der zentrale Behauptung des Satzes ist, daß $(\Delta^{nf}, (\equiv_2 \cup \overset{s}{\Rightarrow}_\alpha)^*)$ eine Definitionsbasis von $(\Delta, \equiv)$ ist. Wir zeigen zuerst, daß Δ^{nf} dicht Δ bezüglich $\equiv$ liegt. Nach Satz 3.4.1 liegt Δ^α dicht in Δ bezüglich $\equiv$. Zu einem beliebigen $E \in \Delta$ existiert somit $F \in \Delta^\alpha$ mit $E \equiv F$. Nach Lemma 5.3.1 ist $[F]$ eine PNF und somit gilt $[F] \in \Delta^{nf}$. Nach Lemma 5.3.4 folgt $F \equiv_1 [F]$ und somit gilt $E \equiv [F]$.

Desweiteren müssen wir nachweisen, daß die Einschränkungen von $\equiv$ und $(\equiv_2 \cup \overset{s}{\Rightarrow}_\alpha)^*$ auf Normalformen übereinstimmen. Dazu seinen $E, F \in \Delta^{nf}$ mit $E \equiv F$. Nach Satz 3.4.1 existiert $G \in \Delta^\alpha$ mit $E \equiv_1 G \overset{*}{}_\alpha\!\overset{s}{\Leftarrow} F$. Wegen Lemma 5.3.3 ist G eine Normalform. Aus Proposition 5.3.5 folgt $[E] \equiv_2 [G]$. Unter Verwendung von Lemma 5.3.2 erhalten wir:

$$E = [E] \equiv_2 [G] = G \overset{*}{}_\alpha\!\overset{s}{\Leftarrow} F\,.$$ □

5.3.2 Reduktionsbasen

In Abbildung 5.6 wiederholen wir die Definition der Relationen $\bar{\rightarrow}_{A_t}$, $\bar{\rightarrow}_{E_t}$ und $\bar{\rightarrow}_{G_t}$. Diese sind hinreichend für Reduktion auf Normalformen im Sinne von Satz 3.4.4, den wir nun beweisen werden.

(A_t)	$\exists\overline{u}(x{:}\overline{y}/F \wedge x\,\overline{z} \wedge E) \;\bar{\to}_{A_t}\; \exists\overline{u}(x{:}\overline{y}/F \wedge F[\overline{z}/\overline{y}] \wedge E)$	
(E_t)	$\exists\overline{u}\exists x(x = y \wedge E) \;\bar{\to}_{E_t}\; \exists\overline{u}\,E[y/x]$	*falls* $y \in \mathcal{V}(\overline{u})$
(G_t)	$\exists\overline{u}\exists\overline{x}(\overline{y}{:}\,\overline{A} \wedge E) \;\bar{\to}_{G_t}\; \exists\overline{u}\,E$	falls $\mathcal{V}(\overline{x}) \cap \mathcal{FV}(E) = \emptyset$, $\mathcal{V}(\overline{y}) \subseteq \mathcal{V}(\overline{x})$ und $\mathcal{V}(\overline{x}) \neq \emptyset$

Abbildung 5.6: Reduktion auf Normalformen

Wir interessieren uns für die Einschränkungen der Axiome (A_t), (E_t) und (G_t) auf $\Delta^{nf} \times \Delta$. Aus diesem Grunde benötigen wir keine Nebenbedingungen, die Kapern von Variablen vermeiden. Desweiteren folgt im Falle von (E_t) die Bedingung $x \neq y$ aus $y \in \mathcal{V}(\overline{u})$ und α-Standardisierung. Letzendlich merken wir an, daß $\bar{\to}_{A_t}$ weder α-Standardisierung noch PNFs erhält.

Für eine Relation R_t auf $\Delta^{nf} \times \Delta$ definieren wir $\overset{2}{\bar{\to}}_{R_t}$ durch $\overset{2}{\bar{\to}}_{R_t} = (\equiv_2 \circ \bar{\to}_{R_t} \circ \equiv)$.

Proposition 5.3.6 *Seien $E \in \Delta^\alpha$, $F \in \Delta$ und R eine der Relationen (A), (E) oder (G). Falls $E \;\bar{\to}_R\; F$ gilt, dann auch $[E] \;\overset{2}{\bar{\to}}_{R_t}\; F$.*

Bevor wir mit dem Beweis dieser Proposition beginnen, zeigen wir, daß wir damit die auf Normalformen basierenden Reduktionsbasen aus Satz 3.4.4 als korrekt beweisen können.

Beweis von Satz 3.4.4. Wegen Satz 3.4.3 liegt Δ^{nf} dicht in Δ bzg. $\equiv$. Wir müssen also lediglich die zweite Bedingung einer Reduktionsbasis überprüfen.

Sei R eine der Relationen (A), (E) oder (G). Wir müssen für alle $E \in \Delta^{nf}$ und $F \in \Delta$, die $E \to_R F$ erfüllen, $E \;\overset{2}{\bar{\to}}_{R_t}\; F$ zeigen. Seien also $E \in \Delta^{nf}$ und $F \in \Delta$ mit $E \to_R F$. Nach Satz 3.4.2 existieren $E_1, F_1 \in \Delta^\alpha$ mit:

$$E \;\equiv_1\; E_1 \;\bar{\to}_R\; F_1 \;\equiv\; F \;.$$

Aus Lemma 5.3.5 folgt $[E] \equiv_2 [E_1]$ und aus Proposition 5.3.6 $[E_1] \;\overset{2}{\bar{\to}}_{R_t}\; F_1$. Mit diesen Eigenschaften und unter Verwendung von Lemma 5.3.2 erhalten wir:

$$E \;=\; [E] \;\equiv_2\; [E_1] \;\overset{2}{\bar{\to}}_{R_t}\; F_1 \;\equiv\; F\,. \qquad \square$$

Wir wenden uns nun dem Beweis von Proposition 5.3.6 zu. Dazu benötigen wir drei neue Begriffe und eine Sammlung von Lemmata. Seien R eine binäre Relation auf Δ und R_t eine Relation auf $\Delta^{nf} \times \Delta$. Das $(R,\, R_t)$ heißt geeignet, wenn es folgende Eigenschaft erfüllt:

$$\text{Falls } \; E \;\bar{\to}_R\; F \; \text{ und } E \in \Delta^\alpha, \text{ dann } \; [E] \;\overset{2}{\bar{\to}}_{R_t}\; F \;.$$

Wir nennen R_t verträglich mit Komposition, wenn für alle E, F und G gilt:

$$[E] \stackrel{2}{\rightarrow}_{R_t} F \text{ und } E \wedge G \in \Delta^{\alpha} \text{ implizieren } [E \wedge G] \stackrel{2}{\rightarrow}_{R_t} F \wedge G \ .$$

Die Relation R_t heißt schwach verträglich mit Komposition, wenn für alle $\overline{u}$, $\overline{v}$, C_1, C_2 und F gilt:

$$\begin{array}{l} \text{Aus } \exists \overline{u} C_1 \stackrel{-}{\rightarrow}_{R_t} F \text{ und } \mathcal{V}(\overline{u}) \cap \mathcal{FV}(C_2) = \emptyset \text{ folgt} \\ \exists \overline{v} \exists \overline{u}(C_1 \wedge C_2) \stackrel{2}{\rightarrow}_{R_t} \exists \overline{v}(F \wedge C_2). \end{array}$$

Lemma 5.3.7 *Sei (R, R_t) ein geeignetes Paar und R_t verträglich mit Komposition und invariant unter Deklaration. Dann implizieren $E \stackrel{-}{\rightarrow}_R F$ und $E \in \Delta^{\alpha}$ auch $[E] \stackrel{2}{\rightarrow}_{R_t} F$.*

Beweis. Mit Indukton über Herleitungen von $E \stackrel{-}{\rightarrow}_R F$. Für den Induktionsanfang verwenden wir, daß das Paar (R, R_t) geeigent ist. Die Betrachtungen für die Regel (*Decl*) sind einfach. Im Falle von Kompositon müssen wir vorsichtig sein, denn die Argumente für die Regeln (*Comp*) und (*Comp'*) sind nicht ganz symmetrisch. Dies sieht man am leichtesten an der Regel (*Comp'*):

$$\frac{E \stackrel{-}{\rightarrow}_R F}{G \wedge E \stackrel{-}{\rightarrow}_R G \wedge F}$$

Dabei gilt $G \wedge E \in \Delta^{\alpha}$. Aus der Induktionsvoraussetzung erhalten wir $[E] \stackrel{2}{\rightarrow}_{R_t} F$. die Verträglichkeit mit Komposition von R_t impliziert $[E \wedge G] \stackrel{2}{\rightarrow}_{R_t} F \wedge G$. Wegen $[E \wedge G] \equiv_2 [G \wedge E]$ und $F \wedge G \equiv G \wedge F$ erhalten wir daraus $[G \wedge E] \stackrel{2}{\rightarrow}_{R_t} G \wedge F$. □

Lemma 5.3.8 *Die Relationen (A_t), (E_t) und (G_t) sind invariant unter Deklaration.*

Beweis. Offensichtlich. □

Lemma 5.3.9 *Eine binäre Relation R_t auf $\Delta^{nf} \times \Delta$ ist verträglich mit Komposition, falls R_t schwach verträglich mit Komposition und sicher ist.*

Beweis. Wir nehmen $\tilde{E} \wedge \tilde{G} \in \Delta^{\alpha}$ und $[\tilde{E}] \stackrel{2}{\rightarrow}_{R_t} \tilde{F}$ an und zeigen $[\tilde{E} \wedge \tilde{G}] \stackrel{2}{\rightarrow}_{R_t} \tilde{F} \wedge \tilde{G}$ zeigen. Nach Definition gibt es E und F mit:

$$[\tilde{E}] \equiv_2 E \stackrel{-}{\rightarrow}_{R_t} F \equiv \tilde{F} \ .$$

Wegen der Lemmata 5.3.1 und 5.3.3 sind E und $[\tilde{G}]$ PNFs und besitzen also Darstellungen:

$$E \;=\; \exists\overline{u}C_1 \qquad \text{und} \qquad [\tilde{G}] \;=\; \exists\overline{v}C_2\,.$$

Aus $\tilde{E} \wedge \tilde{G} \in \Delta^{\alpha}$ folgt $\mathcal{V}(\overline{u}) \cap \mathcal{FV}(C_2) = \emptyset$, so daß wir die schwache Verträglichkeit von R_t mit Komposition ausnützen können:

$$\exists\overline{v}\exists\overline{u}(C_1 \wedge C_2) \;\stackrel{2}{\rightarrow}_{R_t}\; \exists\overline{v}(F \wedge C_2)\,.$$

Als nächstes leiten wir die Eigenschaft $[E \wedge \tilde{G}] \stackrel{2}{\rightarrow}_{R_t} F \wedge \tilde{G}$ her:

$$\begin{array}{rll}
[E \wedge \tilde{G}] & = & \exists\overline{u}\exists\overline{v}(C_1 \wedge C_2) \\
& \equiv_{Exch} & \exists\overline{v}\exists\overline{u}(C_1 \wedge C_2) \\
& \stackrel{2}{\rightarrow}_{R_t} & \exists\overline{v}(F \wedge C_2) \\
& \equiv_{ACI} & \exists\overline{v}(C_2 \wedge F) \\
& \equiv_{Scope} & (\exists\overline{v}C_2) \wedge F \qquad \text{da } \mathcal{V}(\overline{v}) \cap \mathcal{FV}(F) = \emptyset \\
& \equiv_{ACI} & F \wedge [\tilde{G}] \\
& \equiv_1 & F \wedge \tilde{G} \qquad \text{Lemma 5.3.4}
\end{array}$$

Die benötigte Nebenbedingung $\mathcal{V}(\overline{v}) \cap \mathcal{FV}(F) = \emptyset$ folgt aus $\tilde{E} \wedge \tilde{G} \in \Delta^{\alpha}$ und der Sicherheit von R_t:

$$\begin{array}{rcccccc}
\mathcal{FV}(F) & \subseteq & \mathcal{FV}(E) & = & \mathcal{FV}(\tilde{E})\,, & & \\
\mathcal{V}(\overline{v}) & \subseteq & \mathcal{BV}(\exists\overline{v}C_2) & = & \mathcal{BV}([\tilde{G}]) & = & \mathcal{BV}(\tilde{G})\,.
\end{array}$$

Um gesuchte Behauptung sicherzustellen reicht es nun aus, die folgende Eigenschaft nachzuweisen:

$$[\tilde{E} \wedge \tilde{G}] \;\equiv_2\; [E \wedge \tilde{G}]\,.$$

Aus der Idempotenz der Funktion $[\,\_\,]$ (Lemma 5.3.2) folgt:

$$[\tilde{E} \wedge \tilde{G}] \;=\; [[\tilde{E}] \wedge \tilde{G}]\,.$$

Somit ist die zu zeigende Eigenschaft äquivalent zu der nachstehenden:

$$[[\tilde{E}] \wedge \tilde{G}] \;\equiv_2\; [E \wedge \tilde{G}]\,.$$

Wegen $[\tilde{E}] \equiv_2 E$ können wir diese Beziehung durch das allgemeinere Lemma 5.3.10 sicherstellen. □

Lemma 5.3.10 *Aus* $E_1 \equiv_1 E_2$ *folgt* $[E_1 \wedge F] \equiv_2 [E_2 \wedge F]$.

Beweis. Wegen Proposition 5.3.5 und $E_1 \equiv_1 E_2$ gilt $[E_1] \equiv_2 [E_2]$. Nun sind $[E_1]$, $[E_2]$ und $[F]$ nach Proposition 5.3.1 PNFs und haben somit Darstellungen:

$$[E_1] = \exists\overline{u_1}C_1, \qquad [E_2] = \exists\overline{u_2}C_2, \qquad [F] = \exists\overline{v}C\,.$$

Aus einer Anwendung von Proposition 3.5.1 folgt dann:

$$\exists\overline{u_1} \approx \exists\overline{u_2} \qquad \text{und} \qquad C_1 \approx C_2\,,$$

so daß wir $\exists\overline{u_1}\exists\overline{v} \approx \exists\overline{u_2}\exists\overline{v}$ und $C_1 \wedge C \approx C_2 \wedge C$ schließen können. Ein weitere Anwendung von Proposition 3.5.1 in umgekehrter Richtung ergibt:

$$[E_1 \wedge F] \;=\; \exists\overline{u_1}\exists\overline{v}(C_1 \wedge C) \;\equiv_2\; \exists\overline{u_2}\exists\overline{v}(C_2 \wedge C) \;=\; [E_2 \wedge F]\,.$$

□

Lemma 5.3.11 *Die Relationen (A_t), (E_t) und (G_t) sind sicher.*

Beweis. Einfach. □

Lemma 5.3.12 *Das Paar $(A,\, A_t)$ ist geeignet.*

Beweis. Wir setzen $\tilde{E} \in \Delta^\alpha$ und $\tilde{F} \in \Delta$ mit der folgenden Eigenschaft voraus:

$$\tilde{E} \;=\; x{:}\overline{y}/E \wedge x\,\overline{z} \;\;\bar{\rightarrow}_A\;\; x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}] \;=\; \tilde{F}$$

Aus Lemma 5.3.4 erhalten wir $[E] \equiv_1 E$. Nun ist $\equiv_1$ invariant unter Substitution (Proposition 5.2.13), so daß auch $[E][\overline{z}/\overline{y}] \equiv_1 E[\overline{z}/\overline{y}]$ gilt. Die Behauptung erhalten wir nun wie folgt:

$$\begin{array}{rcl}
[\tilde{E}] & = & x{:}\overline{y}/[E] \wedge x\,\overline{z} \\
& \equiv_{ACI} & x{:}\overline{y}/[E] \wedge x\,\overline{z} \;\wedge \top \\
& \bar{\rightarrow}_{A_t} & x{:}\overline{y}/[E] \wedge [E][\overline{z}/\overline{y}] \wedge \top \\
& \equiv_1 & x{:}\overline{y}/E \wedge E[\overline{z}/\overline{y}] \\
& = & \tilde{F}
\end{array}$$

□

Lemma 5.3.13 *Die Paare $(E,\, E_t)$ und $(G,\, G_t)$ sind geeignet.*

Beweis. Überlassen wir dem Leser. □

Lemma 5.3.14 *Die Relation (A_t) ist schwach verträglich mit Komposition.*

Beweis. Seien $\overline{u}$, $\overline{v}$, C_1, C_2 und F gegeben mit $\exists\overline{u}C_1 \stackrel{-}{\rightharpoonup}_{A_t} F$ und $\mathcal{V}(\overline{u}) \cap \mathcal{FV}(C_2) = \emptyset$. Nach Definition von (A_t) existieren x, $\overline{y}$, $\overline{z}$, E_1 und E_2:

$$\exists\overline{u}C_1 \;=\; \exists\overline{u}(x{:}\overline{y}/E_1 \wedge x\,\overline{z} \wedge E_2) \;\stackrel{-}{\rightharpoonup}_{A_t}\; \exists\overline{u}(x{:}\overline{y}/E_1 \wedge E_1[\overline{z}/\overline{y}] \wedge E_2) \;=\; F\,.$$

Die Behauptung können wir nun wie folgt schließen:

$$\begin{array}{rll}
\exists\overline{v}\exists\overline{u}(C_1 \wedge C_2) & \equiv_{ACI} & \exists\overline{v}\exists\overline{u}(x{:}\overline{y}/E_1 \wedge x\,\overline{z} \wedge (E_2 \wedge C_2)) \\
& \stackrel{-}{\rightharpoonup}_{A_t} & \exists\overline{v}\exists\overline{u}(x{:}\overline{y}/E_1 \wedge E_1[\overline{z}/\overline{y}] \wedge (E_2 \wedge C_2)) \\
& \equiv_{ACI} & \exists\overline{v}\exists\overline{u}((x{:}\overline{y}/E_1 \wedge E_1[\overline{z}/\overline{y}] \wedge E_2) \wedge C_2) \\
& \equiv_{Scope} & \exists\overline{v}(\exists\overline{u}(x{:}\overline{y}/E_1 \wedge E_1[\overline{z}/\overline{y}] \wedge E_2) \wedge C_2) \\
& = & \exists\overline{v}(F \wedge C_2)
\end{array}$$

Für die Anwendung von $\equiv_{Scope}$ verwenden wir die Bedingung $\mathcal{V}(\overline{u}) \cap \mathcal{FV}(C_2) = \emptyset$. □

Lemma 5.3.15 *Die Relationen (E_t) und (G_t) sind schwach verträglich mit Komposition.*

Beweis. Überlassen wir wiederum dem Leser. □

Beweis von Proposition 5.3.6. Durch Kombination der Lemmata 5.3.7 bis 5.3.15. □

Literaturverzeichnis

[ACCL90] Martín Abadi, Luca Cadelli, P.-L. Curien, and Jean-Jaques Lévy. Explicit substitutions. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 31–46. ACM Press, 1990.

[ACCL91] Martín Abadi, Luca Cadelli, P.-L. Curien, and Jean-Jaques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991. Also appeared as [ACCL90].

[AFM+95] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of the ACM Symposium on Principles of Programming Languages*. ACM Press, 1995. to appear.

[AK91] Hassan Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. Logic Programming. The MIT Press, Cambridge, Massachusetts, 1991.

[AK93] Hassan Aït-Kaci. An introduction to LIFE — programming with logic, inheritance, functions and equations. In Dale Miller, editor, *Proceedings of the International Symposium on Logic Programming*, pages 52–68. The MIT Press, October 1993.

[AKDM+94] Hassan Aït-Kaci, Bruno Dumant, Richard Meyer, Andreas Podelski, and Peter Van Roy. The Wild LIFE handbook, 1994. Prepublication edition available as part of the Wild LIFE release. The release is available by anonymous ftp from gatekeeper.dec.com in pub/plan/Life1.01.tar.Z.

[AKPS92] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. In *Proceedings of the 5th International Conference on Fifth Generation Computer Systems*, pages 1012–1022, June 1992.

[Bar84] Henk P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier/North Holland, Amsterdam - New York - Oxford, 1984.

[Bar90] H.P. Barendregt. *Functional Programming and Lambda Calculus*, chapter 7, pages 321–363. Volume B of Leeuwen [Lee90], 1990.

[BB90] Gérard Berry and Gérard Boudol. The chemical abstract machine. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 81–94. ACM Press, 1990.

[BMT92] Dave Berry, Robin Milner, and David N. Turner. A semantics for ML concurrency primitives. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 119–129, January 1992.

[BNA91] Paul S. Barth, Rishiyur S. Nikhil, and Arvind. M-structures: Extending a parallel, non-strict, functional language with state. In John Hughes, editor, *Functional Programming Languages and Computer Architecture - 5th ACM Conference*, number 523 in Lecture Notes in Computer Science, pages 538–568. Springer-Verlag, August 1991.

[Bou89] Gérard Boudol. Towards a λ-calculus for concurrent and communicating systems. In *Theory and Practice in Software Development*, number 351 in Lecture Notes in Computer Science, pages 149–161. Springer-Verlag, October 1989.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA, Sophia Antipolis, France, May 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4^{th} Annual ACM SIGPLAN-SIGACT Symposium on Artificial Intelligence & Programming Languages*, pages 238–252. ACM SIGPLAN, 1977.

[CG81] Keith L. Clark and Steve Gregory. A relational language for parallel programming. In *Proceedings of the ACM Conference on Functional Programming Languages and Computer Architecture*, pages 171–178, 1981.

[Chu36] Alonzo Church. An unresolvable problem of number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[CKvC83] Alain Colmerauer, H. Kanoui, and M. van Caneghem. Prolog, theoretical principles and current trends. *Technology and Science of Informatics*, 2(4):255–292, 1983.

[Col90] Alan Colmerauer. An introduction to PROLOG III. *Communications of the ACM*, pages 70–90, July 1990.

[CR91] William Clinger and Jonathan Rees. The Revised[4] Report on the Algorithmic Language Scheme. *LISP Pointers*, IV(3):1–55, July-September 1991. Contributors: Abelson, H. and Adams IV, N.I. and Bartley, D.H. and Brooks, G. and Dybvig, R.K. and Friedman, D.P. and Halstead, R. and Hanson, C. and Haynes, C.T. and Kohlbecker, E. and Oxley, D. and Pitman, K.M. and Rozas, G.J. and Stelle Jr., G.L. and Sussman, G.J. and Wand, M.

[dBKP92] Frank S. de Boer, Jon W. Klop, and Cartuscia Palamidessi. Asynchronous communication in process algebra. In *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 137–147. IEEE Computer Society Press, June 22-25 1992.

[dBP92] Frank S. de Boer and Catuscia Palamidessi. A process algebra of concurrent constraint programming. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 463–477, Cambridge, Massachusetts, 1992. The MIT Press.

[DC93] Daniel Diaz and Philippe Codognet. A minimal extension of the WAM for clp(FD). In *Proceedings of the International Conference on Logic Programming*, pages 774–790. The MIT Press, 1993.

[DH92] Olivier Danvy and John Hatcliff. Thunks (continued). In *In Proceedings of the Workshop on Static Analysis WSA'92*, volume 81-82 of *Bigre Journal*, pages 3–11, September 1992. Bordeaux, France. Also appeared as Technical Report CIS-93-15 of the Kansas State University.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. *Rewrite Systems*, chapter 6, pages 243–320. Volume B of Leeuwen [Lee90], 1990.

[DSVH90] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, 8(1-2):74–94, January-March 1990.

[EHN+93] Marko van Eekelen, Halbe Huitema, Eric Nöcker, Sjaak Smetsers, and Rinus Plasmeijer. *Concurrent Clean, Language Manual.* Katholic University of Nijmegen, April 1993. Version 0.8.4. The language and the manual are available by anonymous ftp from ftp.cs.kun.nl in /pub/Clean.

[EN86] Uffe Engbert and Mogen Nielsen. A calculus of communication systems with label passing. Technical Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1990.

[Frü93] Thom Frühwirth. User-defined constraint handling. In David S. Warren, editor, *Proceedings of the 10th International Conference on Logic Programming*, pages 837–838. The MIT Press, June 1993.

[GAL92] Georges Gonthier, Martín Abadi, and Jean-Jaques Lévy. The geometry of optimal lambda reduction. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 15–26. ACM Press, 1992.

[GS90] Carl A. Gunter and D.S. Scott. *Semantic Domains*, chapter 12, pages 633–674. Volume B of Leeuwen [Lee90], 1990.

[HM94] Martin Henz and Martin Müller. Programming in Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[HMM86] Robert Harper, Dave MacQueen, and Robin Milner. Standard ML. Report ecs-lfcs-86-2, Department of Computer Science, University of Edinburgh, 1986.

[HS88] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, October 1988.

[HSW93] Martin Henz, Gert Smolka, and Jörg Würtz. Oz—a programming language for multi-agent systems. In Ruzena Bajcsy, editor, *13th International Joint Conference on Artificial Intelligence*, volume 1, pages 404–409. Morgan Kaufmann Publishers, 30 August-3 September 1993.

[HSW95] Martin Henz, Gert Smolka, and Jörg Würtz. Object-oriented concurrent constraint programming in Oz. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, chapter 2, pages 27–48. The MIT Press, Cambridge, Massachusetts, 1995. To appear. A previous version is published as [HSW93].

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.

[HWA+90] Paul Hudak, Philip Wadler, Arvind, B. Boutel, J. Fairbairn, J. Fasel, Kevin Hammond, John Hughes, T. Johnsson, D Kieburtz, Rishiyur S. Nikhil, Simon Peyton Jones, M. Reeve, David Wise, and J. Young. Report on the programming language Haskell. Technical Report YALEU/DCS/RR/777, Yale University, Department of Computer Science, 1990.

[HY93] Kohei Honda and Nobuko Yoshida. On reduction-based semantics. In R. K. Shyamasundar, editor, *13^{th} Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India, December 1993.

[Jan94] Sverker Janson. *AKL - A Multiparadigm Programming Language.* PhD thesis, SICS Swedish Institute of Computer Science, SICS Box 1263, S-164 28 Kista, Sweden, 1994. SICS Dissertation Series 14.

[JH91] Sverker Janson and Seif Haridi. Programming paradigms of the Andorra Kernel Language. In Vijay Saraswat and Kazunori Ueda, editors, *Proceedings of the 1991 International Symposium on Logic Programming*, pages 167–186, San Diego, California, October 1991.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 111–119. ACM Press, 1987.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–582, May-July 1994.

[JMH93] Sverker Janson, Johan Montelius, and Seif Haridi. Ports for objects. In *Research Directions in Concurrent Object-Oriented Programming.* The MIT Press, Cambridge, Massachusetts, 1993.

[Kat90] Vinod Kathail. *Optimal Interpreters for Lambda-Calculus based Functional Languages*. PhD thesis, Departement of Electrical Engeneering and Computer Science, MIT, 1990.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 16–30. ACM Press, 1990.

[Lau93] John Launchbury. A natural semantics for lazy evaluation. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 144–154. ACM Press, 1993.

[Lee90] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume B. The MIT Press, Cambridge, Massachusetts, 1990.

[Les94] Pierre Lescanne. From $\lambda\sigma$ to $\lambda\nu$ a journay through calculi of explicit substitutions. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 60–69. ACM Press, June 1994.

[LMM88] Jean-Louis Lassez, Michael J. Maher, and Kim Marriott. Unification revisited. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 15, pages 587–625. Morgan Kaufman Publisher Inc., Los Altos, California, 1988.

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall Software Series. Prentice Hall International Editions, Englewood Cliffs, New Jersey, 1981.

[LTLG92] Jean-Jaques Lévy, Bent Thomsen, Lone Leth, and Alessandro Giacalone. Concurrency and functions: Evaluation and reduction. In *EATOS*. ESPRIT Basic Research Action 6454-CONFER, November 1992.

[Mah87] Michael J. Maher. Logic semantics for a class of committed-choice programs. In Jean-Louis Lassez, editor, *Logic Programming, Proceedings of the Fourth International Conference*, pages 858–876. The MIT Press, 1987.

[Mar90] Luc Maranget. Optimal derivations in weak lambda-calculi and in orthogonal term rewriting systems. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 255–269. ACM Press, 1990.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, London, 1989.

[Mil90] Robin Milner. *Operational and Algebraic Semantics of Concurrent Processes*, chapter 19, pages 1201–1242. Volume B of Leeuwen [Lee90], 1990.

[Mil91] Robin Milner. The polyadic π-calculus: A tutorial. ECS-LFCS Report Series 91–180, Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, October 1991.

[Mil92] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[MMPS94] Michael Mehl, Tobias Müller, Konstantin Popow, and Ralf Scheidhauer. The DFKI Oz user's manual. Research Report RR-94-29, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[MN95] Martin Müller and Joachim Niehren. Higher-order meta programming in Oz. In Preparation, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, {niehren,mmueller}@dfki.uni-sb.de, 1995.

[Mos90] P.D. Mosses. *Denotational Semantics*, chapter 11, pages 575–631. Volume B of Leeuwen [Lee90], 1990.

[MPSW94] Tobias Müller, Konstantin Popow, Christian Schulte, and Jörg Würtz. Constraint programming in Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Journal of Information and Computation*, 100:1–77, 1992.

[MS95] Martin Müller and Gert Smolka. The polymorphically typed gamma-calculus. In Preparation, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, mmueller@dfki.uni-sb.de, 1995.

[Mül94] Martin Müller. A constraint-based recast of ML-polymorphism. In Denis Lugiez, editor, *8^{th} Internationl Workshop on Unification*, June 1994. Extended Abstract, to appear.

[Nik91] Rishiyur S. Nikhil. *Id Language Reference Manual (Version 90.1)*. Computation Structures Group - Laboratory for Computer Science - Massachusetts institute of Technology, 545 Technology Square, Cambridge Massachusetts 02139, USA, July 1991.

[Nik94] Rishiyur S. Nikhil. An overview of the parallel language Id - a foundation for pH, a parallel dialect of Haskell. Technical report, Digital Cambridge Research Laboratory, 1994. Submitted for Publication nikhil@crl.dec.com.

[NM88] Gopalan Nadathur and Dale Miller. An overview of λ-Prolog. In Kevin Bowen and Robert Kowalski, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 810–827. The MIT Press, 1988.

[NP93] Joachim Niehren and Andreas Podelski. Feature automata and recognizable sets of feature trees. In *Theory and Practice of Software Development, 4th International Joint Conference CAAP/FASE*, volume 668 of *Lecture Notes in Computer Science*, pages 356–375. Springer-Verlag, April 1993.

[NPT93] Joachim Niehren, Andreas Podelski, and Ralf Treinen. Equational and membership constraints for infinite trees. In *Proceedings of the Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 106–120. Springer-Verlag, June 1993.

[NS94] Joachim Niehren and Gert Smolka. A confluent calculus for higher-order relational programming with constraints. In *Proceedings of the First International Conference on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*. Springer-Verlag, September 1994.

[OD93] Gerald K. Ostheimer and Anthony J. K. Davie. π-calculus characterizations of some practical λ-calculus reduction strategies. Technical report, University of St. Andrews, Scotland, October 1993.

[Ode94] Martin Odersky. A functional theory of local names. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 48–59. ACM Press, January 1994.

[O'K90] Richard A. O'Keefe. *The Craft of Prolog*. The MIT Press, Cambridge, Massachusetts, 1990.

[ORH93a] Martin Odersky, Dan Rabin, and Paul Hudak. Call by name, assignment, and the lambda calculus. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 43–56. ACM Press, January 1993. A nice complete version is published as [ORH93b].

[ORH93b] Martin Odersky, Dan Rabin, and Paul Hudak. The unexpurgated call by name, assignment, and the lambda calculus. Research Report RR-94-04, YALE University/ DCS, Dept. of Computer Science, Yale University, March 1993.

[PJ87] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages.* International Series in Computer Science. Prentice-Hall, 1987.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Journal of Theoretical Computer Science*, 1:125–159, 1975.

[PS92] S. Purushothaman and Jill Seaman. An adequate operational semantics of sharing in lazy evaluation. In *Europeen Symposium on Programming (ESOP)*, volume 582 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[PS93] Andrew Pitts and Ian Stark. On the observable properties of higher order functions that dynamically create local names. In *Proceedings of the ACM SIGPLAN Workshop on State in Programming Languages*, pages 31–45, June 1993.

[San92] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[Sar89] Vijay A. Saraswat. *Concurrent Constraint Programming Languages.* PhD thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, 1989.

[Sar91] Vijay A. Saraswat. Semantic foundation of concurrent constraint programming. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 333–352. ACM Press, 1991.

[Sha87] Ehud Shapiro, editor. *Concurrent Prolog. Collected Papers. Volume 1 and 2.* The MIT Press, Cambridge, Massachusetts, 1987.

[Smo94a] Gert Smolka. A calculus for higher-order concurrent constraint programming with deep guards. Research Report RR-94-03, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, February 1994.

[Smo94b] Gert Smolka. The definition of Kernel Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, November 1994.

[Smo94c] Gert Smolka. A foundation for concurrent constraint programming. In *Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 50–72. Springer-Verlag, September 1994.

[Smo94d] Gert Smolka. An Oz primer. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[SR90] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 232–245. ACM Press, January 1990.

[SS86] Leon Sterling and Ehud Shapiro. *The Art of Prolog.* The MIT Press, Cambridge, Massachusetts, 1986.

[SS94] Christian Schulte and Gert Smolka. Encapsulated search in higher-order concurrent constraint programming. In Maurice Bruynooghe, editor, *Logic Programming: Proceedings of the 1994 International Symposium*, pages 505–520. The MIT Press, November 1994.

[SSW94] Christian Schulte, Gert Smolka, and Jörg Würtz. Encapsulated search and constraint programming in Oz. In A.H. Borning, editor, *Second Workshop on Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*, pages 134–150. Springer-Verlag, 2-4 May 1994.

[ST92] Gert Smolka and Ralf Treinen. Records for logic programming. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 240–254. The MIT Press, 1992.

[Tho89] Bent Thomsen. A calculus of higher-order communicating systems. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 142–154. ACM Press, 1989.

[TLP+93] Bent Thomsen, Lone Leth, Sanjiva Prasad, Tsung-Min Kuo, Andre Kramer, Fritz Knabe, and Alessandro Giacalone. Facile Antigua release programming guide. Technical Report ECRC-93-20, ECRC, European Computer-Industry Research Centre, ECRC GmbH, Arabellastr. 17, D-81925 Munich, December 1993.

[Tur85] David A. Turner. Miranda: A non-strict functional language with polymorphic types. In Jean-Pierre Jouannaud, editor, *IFIP International Conference on Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1985.

[Vas94a] Vasco T. Vasconcelos. *A Process Calculus Approach to Typed Concurrent Objects*. PhD thesis, Keio University, June 1994.

[Vas94b] Vasco T. Vasconcelos. Typed concurrent objects. In *8th Proceedings of the European Conference on Object Oriented Programming*, volume 821 of *Lecture Notes in Computer Science*, pages 100–117. Springer-Verlag, July 1994.

[vHSD93] Pascal van Hentenryck, Vijay A. Saraswat, and Yves Deville. Design, implementation and evaluation of the constraint language cc(FD). Report CS-93-02, Brown University, January 1993.

[VT93] Vasco T. Vasconcelos and Mario Tokoro. A typing system for a calculus of objects. In *ISOTAS*, number 742 in Lecture Notes in Computer Science, pages 460–574. Springer-Verlag, November 1993.

[Wad71] C. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, Oxford, United Kingdom, September 1971.

[Wal91] David Walker. π-calculus semantics of object-oriented programming languages. In T. Ito and A. R. Meyer, editors, *Proceedings of International Conference on Theoretical Aspects of Computer Software (TACS'91)*, volume 526 of *Lecture Notes in Computer Science*, pages 532–547. Springer-Verlag, September 1991.

[War80] D. H. D. Warren. Logic programming and compiler writing. *Software-Practice and Experience*, 10(2), 1980.

[WH81] Patrick H. Winston and Berthold K. Horn. *LISP*. Addison-Wesley, 1981.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages.* The MIT Press, Cambridge, Massachusetts, 1993.

[Yos93] Nobuko Yoshida. Optimal reduction in weak λ-calculus with shared environments. In *ACM Conference on Functional Programming Languages and Computer Architecture*, 1993.

Stichwortverzeichnis

Symbolverzeichnis