Vergleich von Narrowing-Verfahren erster und höherer Ordnung

Alexander Pretschner

Diplomarbeit

Informatik

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinisch-Westfälischen Technischen Hochschule Aachen
im Juni 1998

Angefertigt am
Lehr- und Forschungsgebiet Informatik II
Prof. Dr. Michael Hanus

Ich versichere, die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Aachen, den 15. Juni 1998

Inhaltsverzeichnis

1 Einleitung **3**

1.1 Zielsetzung und Gliederung 4

1.2 Notationen und Definitionen 6

2 Narrowing erster Ordnung **9**

2.1 Needed Narrowing 10

2.2 OINC 14

2.3 Strikte Gleichheit 17

2.4 s-OINC 20

2.5 Diskussion 24

3 Systematischer Vergleich von OINC und NN **25**

3.1 Motivation: LR-definierende Bäume 26

3.2 Vergleich einzelner NN- und OINC-Schritte 29

3.3 Vergleich erfolgreicher Ableitungen 38

3.3.1 Simulation von NN durch OINC 39

3.3.2 Simulation von OINC durch NN 57

3.4 Vergleich der Komplexitäten 67

3.4.1 Grundsätzliche Überlegungen 68

3.4.2 Experimentelle Resultate 72

3.5 Simulation von s-OINC und NN 76

3.5.1 Vergleich der Komplexitäten: Grundsätzliches 82

3.5.2 Vergleich der Komplexitäten: Experimentelle Resultate . . 83

3.6 Diskussion 84

4 Narrowing höherer Ordnung **86**

4.1 Warren's Methode 87

4.2 Applikative TES 94

4.3 Eine Erweiterung von OINC: NCA 96

4.4 Simulation von NN durch NCA 99

4.5 Zur Simulation von NCA durch NN 109

4.6 Strikte Gleichheit: s-NCA 122

4.7 Vergleich der Komplexitäten 122

4.8 Diskussion 127

5 Implementierung 130

5.1 NN 130

5.2 OINC 133

5.3 s-OINC 134

5.4 NCA 136

5.5 Diskussion 140

6 Zusammenfassung und Ausblick 142

6.1 Narrowing erster Ordnung 142

6.2 Narrowing höherer Ordnung 144

Abkürzungsverzeichnis 147

Symbolverzeichnis 148

Tabellenverzeichnis 150

Literaturverzeichnis 151

Kapitel 1

Einleitung

Das Paradigma der deklarativen Programmierung läßt sich vereinfachend dahingehend charakterisieren, daß für ein gegebenes Problem nicht die *Lösung* bzw. der *Lösungsweg* implementiert, sondern das *Problem* als solches beschrieben wird. Die Berechnung einer Lösung wird dann unter Rückgriff auf allgemeine Problemlösungsstrategien (Termersetzung, Resolution) durchgeführt.

Die deklarative Programmierung ermöglicht somit das Schreiben – im Vergleich zur imperativen Programmierung – kürzerer und damit besser wartbarer Programme. Weitere Vorteile sind die referentielle Transparenz und damit verknüpft die Abwesenheit – oder *kontrollierte* Anwesenheit – von Seiteneffekten. Allerdings führt die Verwendung deklarativer Programmiersprachen allein natürlich nicht dazu, daß "bessere" Software erstellt wird. Dazu sind sorgfältige Überlegungen zum Design eines Systems mit Methoden aus der Softwaretechnik anzustellen [Nag90]. Deklarative Sprachen können nur helfen, den Prozeß der Softwareentwicklung effizienter und überschaubarer zu gestalten.

Das Hauptanwendungsgebiet der deklarativen Programmierung ist nach wie vor fast ausschließlich die Künstliche Intelligenz. Das häufig vorgebrachte Argument, deklarative Sprachen seien zu langsam, überzeugt angesichts immer ausgefeilterer Implementierungen und der rasanten Entwicklung immer schnellerer Prozessoren nicht mehr.
Außerdem ist davon auszugehen, daß die im Detail durchaus anspruchsvolle deklarative Denkweise sich stärker durchsetzen wird, wenn Programmierer früher und intensiver als bisher mit ihr konfrontiert werden. Wenn jahrelang in einer imperativen Sprache programmiert wurde, ist der Schritt in eine neue Denkweise ein sehr großer. Diese Erfahrung macht jeder, der das erste Mal ein objektorientiertes Programm zu schreiben versucht.

Es gibt zwei große Klassen von deklarativen Programmiersprachen: die logischen und die funktionalen Sprachen.
Die logischen Sprachen einerseits (mit dem Hauptvertreter PROLOG) basieren auf einer Einschränkung der Prädikatenlogik erster Ordnung, der Hornklausellogik.

Die funktionalen Sprachen andererseits (Haskell, Miranda, Gofer, Lisp) basieren auf der Vorstellung, Probleme als i.a. rekursive Funktionen notieren zu können. Funktionale Programme bestehen dann aus einer Menge solcher Funktionsdefinitionen.

1.1 Zielsetzung und Gliederung

In den letzten fünfzehn Jahren wurde nun versucht, die funktionalen und die logischen Sprachen zu einem neuen Konzept zu integrieren: den funktional-logischen Sprachen, die die Vorteile beider Sprachklassen in sich vereinigen (Reduktion geschachtelter Ausdrücke, lazy Auswertung, Funktionen höherer Ordnung auf Seiten der funktionalen Programmierung und logische Variablen, partielle Datenstrukturen und in die Sprache integrierte Suchmechanismen auf Seiten der Logikprogrammierung; [Han97]).
Die zugrundeliegende operationelle Semantik dieser Sprachen ist Residuation (Escher [Llo94], [Llo95], Le Fun [Aït90], Life [ALN87], NUE-Prolog [Nai91], Oz [Smo95]) oder Narrowing (ALF [Han90], Babel [MR92], K-Leaf [GLMP91], LPG [BE86], ROSE [BBA96], SLOG [Fri85], TOY [GHLR96]).
Eine Integration beider Mechanismen [Han97] findet in der Sprache Curry ([HKM95], [Han98]) statt.

Diese Arbeit besteht aus zwei Teilen. Im ersten werden zwei Narrowing-Verfahren erster Ordnung gegenübergestellt und ausführlich miteinander verglichen. Der zweite Teil beschäftigt sich mit einem Aspekt funktional-logischer Sprachen, der zentral in der funktionalen Programmierung ist. Es handelt sich darum, Funktionen als “first class citizens” zu behandeln, d.h. sie als Argumente oder Rückgabewerte von Funktionen zuzulassen (Funktionen höherer Ordnung). In funktional-logischen Sprachen ist es insbesondere wünschenswert, Instantiierungen von logischen Variablen mit (partiell applizierten) Funktionen zu gestatten und berechnen zu können und somit das Konzept der Funktionen als “first class citizens” auf funktional-logische Sprachen zu erweitern.
Dazu bieten sich zwei Ansätze an: Entweder implementiert man Narrowing höherer Ordnung mit Unifikation höherer Ordnung ([HP96], [Pre95a], [Qia94], HO-Babel [KA96], [Kuc95], λ-Prolog [NM88]), oder man verzichtet auf die Unifikation höherer Ordnung. Ansätze dazu sind die Verwendung applikativer Termersetzungssysteme ([NMI95]; konditionale applikative TES in SFL [GHR92] und in TOY mit einer Ersetzungslogik [GHR97]) oder eine im Bereich der Logikprogrammierung von D.H.D. Warren vorgeschlagene syntaktische Transformation ([War82], [CER90], [Gon93], [Rey72]; IDEAL [BG86] als Ergänzung von K-Leaf [GLMP91] um Konstrukte höherer Ordnung; Berücksichtung von Typen für monomorphe Programme in [AT98] und bei polymorphen Programmen im funktionalen Kontext in [BBH97]). Die Idee der Warren'schen Transformation ist es, partiell applizierte Funktionen oder Konstruktoren durch neue (parametrisierte) Konstruktoren zu repräsentieren.

Die in der vorliegenden Arbeit untersuchten Kalküle umgehen die Unifikation höherer Ordnung. Deshalb werden zunächst zwei Kalküle erster Ordnung, nämlich der Outside-In-Narrowing-Calculus (OINC, [IN97]) und Needed Narrowing (NN, [AEH94a]) verglichen. Eine Weiterentwicklung von OINC, der Narrowing Calculus for Applicative Systems (NCA, [NMI95]) wird dann Needed Narrowing in Kombination mit einer Implementierung der Warren'schen Methode gegenübergestellt.

Es wird gelingen, eine eineindeutige Zuordnung zwischen den von NN und OINC errechneten erfolgreichen Ableitungen vorzunehmen. Diese Zuordnung ist konstruktiv, d.h. es wird gezeigt, wie die sich entsprechenden Ableitungen aussehen. Der Beweis erfolgt über eine direkte Zuordnung von NN- zu OINC-Ableitungen (erfolgreich oder fehlschlagend) und den Nachweis, daß die von OINC und NN berechneten Lösungen identisch sind (Einschränkung auf erfolgreiche Ableitungen).
Diese gegenseitige Simulationsbeziehung bedeutet allerdings nicht, daß sich beide Kalküle in einer Implementierung auch gleich verhalten. Es wird sich herausstellen, daß OINC's nichtdeterministisches Verhalten bei der Auswahl einer passenden Regel früher (in bezug auf die Länge der Ableitung) als in NN zutage tritt und deshalb mehrere – später fehlschlagende – Ableitungen betrachtet werden müssen. Damit ergibt sich insbesondere, daß OINC im Gegensatz zu NN nicht deterministisch auf Grundtermen arbeitet.
Dieses schlechtere Verhalten wird besonders augenfällig, wenn Programme mit strikter Gleichheit betrachtet werden (NN ist überhaupt nur für strikte Gleichungen korrekt und vollständig). Eine Modifikation von OINC für strikte Gleichungen, s-OINC, wird ebenfalls durch gegenseitige Simulation mit NN in Beziehung gesetzt; auch hier ist das Implementierungsverhalten wegen der ungünstigen Behandlung des auftretenden Nichtdeterminismus "schlechter" als das von NN.

Im Fall der höheren Ordnung ist eine ähnliche konstruktive Zuordnung von NN- zu NCA-Ableitungen möglich. Die Simulation von NN durch NCA wird angegeben. Eine Simulation von NCA durch NN mit Warren's Methode mit den gewählten Methoden i.a. nicht so einfach durchzuführen. Das liegt insbesondere daran, daß die zugrundeliegenden TES unterschiedlich sind. Es wird nachgewiesen, daß für bestimmte dem NCA-Kalkül zugrundeliegende Inferenzregeln "äquivalente" OINC- und damit NN-Ableitungen existieren.
Für die übrigen Regeln wird begründet, warum das Rechenverhalten von NN besser als das von NCA sein wird, allerdings nicht formal nachgewiesen. Diese Argumentation wird mit experimentellen Resultaten untermauert.
Wenn man allerdings partielle Funktionsapplikationen auf rechten Regelseiten verbietet, so kann auch die Simulation erfolgreicher NCA- durch NN-Ableitungen mit einem "Umweg" über OINC nachgewiesen werden.

Dem Verfasser ist keine Arbeit bekannt, die Narrowingkalküle in einer derartigen konstruktiven Weise in Beziehung setzt.

Der Sinn dieser Arbeit liegt neben dem theoretischen Interesse des Vergleichs der Kalküle in einer Argumentationsgrundlage für die Verwendung von NN als die einer funktional-logischen Programmiersprache höherer Ordnung zugrundeliegende operationelle Semantik. Die Schlußfolgerung wird nämlich sein, daß NN im Fall erster und höherer Ordnung weniger Zeit und weniger Speicherplatz für die Errechnung von Lösungen benötigt als OINC bzw. NCA. Da immer ein Tradeoff zwischen der Mächtigkeit der zugrundeliegenden TES und der Effizienz besteht, ist dieses Resultat angesichts der Tatsache, daß NN für induktiv-sequentielle und OINC bzw. NCA für orthogonale Systeme definiert ist, geeignet zu interpretieren. Die Praxis zeigt allerdings, daß die meisten Programme tatsächlich induktiv-sequentiell sind.

Im einzelnen gliedert sich diese Arbeit wie folgt: Nach einer Einführung der verwendeten Notationen werden in Kapitel 2 die Narrowing-Kalküle erster Ordnung NN und OINC vorgestellt. Dabei wird auch auf die strikte Gleichheit eingegangen.
In Kapitel 3 folgt die eindeutige Zuordnung von einzelnen NN-Schritten zu OINC-Ableitungen und dann von vollständigen (erfolgreichen) Ableitungen. Mit dem Nachweis der Identität der von NN und OINC errechneten Lösungsmengen wird dann die wechselseitige Simulationsbeziehung erfolgreicher Ableitungen aufgezeigt. Hier wird der Terminus "Bisimulation" nicht verwendet, weil zwar jeder NN- eine OINC-Ableitung zugeordnet wird (erfolgreich, fehlschlagend oder unendlich), die Simulation von OINC durch NN aber nur für erfolgreiche Ableitungen aufgezeigt wird. Daran schließen sich Überlegungen zur Komplexität der Algorithmen an, die durch experimentelle Resultate ergänzt werden.
Kapitel 4 beginnt mit einer Vorstellung von Warren's Methode, dem Kalkül NCA und den NCA zugrundeliegenden Termersetzungssystemen. Daraufhin wird NN mit Warren's Methode durch NCA simuliert. Die Rückrichtung wird ausführlich diskutiert, auch wenn eine konstruktive Zuordnung nicht für alle Inferenzregeln angegeben wird (die vollständige Zuordnung erfolgt für den Fall, daß auf rechten Regelseiten keine partiellen Funktionen zugelassen werden). Es folgt eine Begründung, warum mit den gewählten Methoden eine konstruktive Zuordnung der Ableitungen nicht ohne weiteres möglich ist sowie der begründete Verdacht, daß trotzdem eine wechselseitige Simulationsbeziehung besteht. Überlegungen zur Komplexität mit ergänzenden experimentellen Resultaten schließen dieses Kapitel ab.
Kapitel 5 stellt die bei der Ermittlung der experimentellen Resultate verwendeten PROLOG-Implementierungen wor.
Kapitel 6 schließt diese Arbeit mit einer Diskussion und einem Ausblick ab. Darin wird u.a. auf die Mächtigkeit (d.h. verwendbare TES) der verwendeten Kalküle eingegangen.

1.2 Notationen und Definitionen

Dieser Abschnitt stellt die verwendeten üblichen Notationen und Definitionen enumerativ vor.

$\Sigma = \mathcal{F} \cup \mathcal{C}$ sei eine Signatur mit (definierten) Funktionssymbolen aus $\mathcal{F}$ und Konstruktorsymbolen aus $\mathcal{C}$, wobei $\mathcal{F} \cap \mathcal{C} = \emptyset$ ist. Für eine Variablenmenge $\mathcal{X}$ sind Terme $\mathcal{T}(\Sigma, \mathcal{X})$ oder nur $\mathcal{T}$ wie üblich definiert. Ein Term $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ heißt Konstruktorterm, im Fall $\mathcal{X} = \emptyset$ Grundkonstruktorterm. $\mathcal{V}ar$ bezeichnet eine Variablenmenge; $\mathcal{V}ar(\mathcal{E})$ die Variablenmenge eines syntaktischen Objekts $\mathcal{E}$. Die Menge $\mathcal{O}(t)$ ist eine Menge von Folgen natürlicher Zahlen (Positionen, durch Punkte $\cdot$ voneinander abgesetzt), die Unterterme eines Terms t adressieren. Die leere Folge wird mit ε bezeichnet. Eine Position $u \in \mathcal{O}(t)$ adressiert den Unterterm $t|_u$, wobei $t|_u = t$ für $u = \varepsilon$ und im Fall $t = f(t_1, \dots, t_n)$ $t|_u = t_i|_{u'}$ für $u = i \cdot u'$ ist. Eine Position $u \in \mathcal{O}(t)$ heißt nichtvariabel, wenn $t|_u \notin \mathcal{V}ar(t)$ ist. Die Menge der nichtvariablen Positionen eines Terms t wird mit $\overline{\mathcal{O}}(t)$ bezeichnet. $t[s]_p$ bezeichnet das Resultat der Ersetzung von $t|_p$ durch s in t. $Head(t)$ ist das Kopfsymbol von t, d.h. für $t = f(t_1, \dots, t_n)$ $(n \geq 0)$ ist $Head(t) = f$, und für $t \in \mathcal{V}ar$ ist $Head(t) = t$. Ein Konstruktorkopfterm ist ein Term t mit $Head(t) \in \mathcal{C}$.
Die partielle Ordnung $\preceq$ über Positionen ist wie folgt definiert: Für $u, v \in \mathcal{O}(t)$ gilt $v \preceq u$, falls v Präfix von u ist, d.h. $\exists w \; vw = u$. $v \prec u$ gilt, falls $v \preceq u$ und $u \neq v$ ist. $\succ$ und $\succeq$ sind entsprechend definiert. Falls nötig, wird $\preceq$ auf Mengen erweitert, d.h. $v \preceq \{u_1, \dots, u_n\}$, falls $v \preceq u_i$ für $1 \leq i \leq n$ gilt. u heißt links von p, falls u als $w_1 \cdot i \cdot w_2$ und v als $w_1 \cdot j \cdot w_3$ mit $i < j$ geschrieben werden kann. Die Menge $Left(t, p)$ bezeichnet die Menge aller Positionen aus t, die links von p liegen. Wenn der Kontext klar ist, wird nur von $Left(p)$ für eine Position p gesprochen. $Right(t, p)$ und $Right(p)$ sind entsprechend definiert. $u \in O (\subseteq \mathcal{O}(t))$ heißt leftmost, falls für kein $v \in O$ $v \in Left(t, u)$ gilt. Eine Position heißt atomar, falls sie nicht als Konkatenation von Positionen geschrieben werden kann (ε eingeschlossen; Konkatenationen mit ε sind gestattet, aber wenig sinnvoll).
Eine Substitution σ ist eine Abbildung von $\mathcal{V}ar$ nach $\mathcal{T}$ mit dem Urbildbereich $dom(\sigma)$, wobei $dom(\sigma) = \{x \in \mathcal{V}ar | \sigma(x) \neq x\}$ endlich ist. Der Bildbereich von σ ist als $cod(\sigma) = \{\sigma(x) | x \in dom(\sigma)\}$ definiert. Die Menge aller Substitutionen ist $Subst$. Die leere Substitution wird als $\emptyset$ oder $\{\}$ notiert. Eine Substitution, deren Bildbereich keine Funktionssymbole enthält, heißt Konstruktorsubstitution. Substitutionen werden wie üblich zu Homomorphismen auf Termen fortgesetzt. Eine Variablenumbenennung ϱ ist eine bijektive Substitution mit $cod(\varrho) \subseteq \mathcal{V}ar$. Ein Term t' heißt Variante eines Terms t, falls es eine Variablenumbenennung ϱ mit $\varrho(t) = t'$ gibt. Die Komposition zweier Substitutionen σ und ϑ wird mit $\sigma \circ \vartheta$ bezeichnet (erst ϑ, dann σ anwenden). Die Subsumtionsbeziehung $\leq$ von Substitutionen ist wie folgt definiert: $\sigma \leq \vartheta$ für $\sigma, \vartheta \in Subst$, falls $\exists \lambda \in Subst \; \lambda \circ \sigma = \vartheta$. σ heißt dann allgemeiner als ϑ. $\sigma \sim \vartheta$ gilt genau dann, wenn $\sigma \leq \vartheta$ und $\vartheta \leq \sigma$ gilt. Im Fall $\sigma \leq \vartheta$ und $\sigma \not\sim \vartheta$ gilt $\sigma < \vartheta$. Im folgenden wird ausschließlich $\sigma = \vartheta$ für $\sigma \sim \vartheta$ verwendet (d.h. insbesondere, daß die syntaktische Gleichheit von Substitutionen nicht verwendet wird, wenn nicht explizit darauf hingewiesen wird). $\sigma \in Subst$ heißt Unifikator von $t, s \in \mathcal{T}$, wenn $\sigma(s) = \sigma(t)$ ist. σ heißt allgemeinster Unifikator (mgu), falls σ t und s unifiziert und es keinen Unifikator $\vartheta < \sigma$ für s und t gibt. $s \leq t$ gilt für $s, t \in \mathcal{T}$, falls es eine Substitution σ mit $\sigma(s) = t$ gibt. t heißt dann Instanz von s. σ und ϑ heißen unvergleichbar bzgl. $\leq$, falls $\sigma \not\leq \vartheta$ und $\vartheta \not\leq \sigma$ ist. σ und ϑ heißen un-

abhängig, falls es ein $x \in \mathcal{V}ar$ gibt, so daß $\sigma(x)$ und $\vartheta(x)$ nicht unifizierbar sind. Die Einschränkung von σ auf die Variablenmenge $\mathcal{X}$ wird mit $\sigma \restriction_{\mathcal{X}}$ bezeichnet. Es ist $\sigma = \vartheta[\mathcal{X}]$, falls $\sigma \restriction_{\mathcal{X}} = \vartheta \restriction_{\mathcal{X}}$ ist. $\sigma \leq \vartheta[\mathcal{X}]$ gilt im Fall $\eta \circ \sigma = \vartheta[\mathcal{X}]$ für eine Substitution η.
Für eine binäre Relation R ist R^+ der transitive und R^* der transitive und reflexive Abschluß von R. Eine Ersetzungsregel ist ein Paar $l \to r$ von Termen mit $l \notin \mathcal{V}ar$ und $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. Ein Termersetzungssystem (TES) $\mathcal{R}$ ist eine Menge von Ersetzungsregeln. Eine Funktion f heißt definierte Funktion (bzgl. $\mathcal{R}$), falls es eine Regel $l \to r \in \mathcal{R}$ mit $Head(l) = f$ gibt. Die Menge der definierten Funktionen wird mit $\mathcal{F}$ bezeichnet. Die restlichen Symbole in den linken und rechten Regelseiten sind Variablen, Klammern oder Konstruktoren ($\mathcal{C}$; s.o.). Die Signatur eines TES besteht aus $\mathcal{F} \cup \mathcal{C}$. Jedem Element f der Signatur ist seine Stelligkeit (intuitiv die Anzahl der Argumente), $Arity(f)$, zugeordnet. Ein TES $\mathcal{R}$ heißt konstruktorbasiert, falls für jede Regel $f(l_1, \dots, l_n) \to r$ $f \in \mathcal{F}$ ist und $l_1, \dots, l_n$ Konstruktorterme sind. Ein Term t heißt linear, wenn jede Variable aus t nur einmal vorkommt. Eine Regel $l \to r$ heißt linkslinear, wenn l linear ist. Ein TES $\mathcal{R}$ heißt orthogonal, wenn jede Regel aus $\mathcal{R}$ linkslinear ist und es für jedes Paar $l \to r$, $l' \to r'$ von Regelvarianten keinen Unifikator von l und $l'|_u$ für alle $u \in \overline{\mathcal{O}}(l')$ gibt, es sei denn, $l \to r$ und $l' \to r'$ sind Varianten derselben Regel, und es ist $u = \varepsilon$. Die durch $\mathcal{R}$ induzierte Ersetzungsrelation $\to_{\mathcal{R}}$ ist durch $s \to_{\mathcal{R}} t \Leftrightarrow \exists p \in \overline{\mathcal{O}}(s) \exists \sigma \in Subst \exists l \to r \in \mathcal{R}\ s|_u = \sigma(l) \wedge t = s[\sigma(r)]_p$ definiert, wobei $l \to r$ eine Regel ist. Wenn der Zusammenhang klar ist, wird der Subskript $\mathcal{R}$ weggelassen. Bei Bedarf erhält $\to$ als Index die Position, die Substitution, die verwendete Regel oder Kombinationen davon. $=_{\mathcal{R}}$ bezeichnet den reflexiven, transitiven und symmetrischen Abschluß von $\to_{\mathcal{R}}$. Für Substitutionen σ und ϑ gilt $\sigma =_{\mathcal{R}} \vartheta$, falls $\forall x \in \mathcal{V}ar\ \sigma(x) =_{\mathcal{R}} \vartheta(x)$ ist, $\sigma \leq_{\mathcal{R}} \vartheta$, falls $\exists \eta \in Subst\ \eta \circ \sigma =_{\mathcal{R}} \vartheta$, und $\sigma \leq_{\mathcal{R}} \vartheta[\mathcal{X}]$, falls $\exists \eta \in Subst\ \eta \circ \sigma =_{\mathcal{R}} \vartheta[\mathcal{X}]$.
$\overrightarrow{x_k}$ bezeichnet das Tupel $x_1, \dots, x_k$. $\overline{x_k}$ bezeichnet das (applikative) Tupel $x_1 \dots x_k$. Eine Unterterm $t|_p$ heißt Redex (bzgl. eines TES $\mathcal{R}$), falls es eine Regel $l \to r \in \mathcal{R}$ und eine Substitution σ mit $\sigma(l) = t|_p$ gibt. Ein Term, der kein Redex besitzt, ist in Normalform. Eine Folge von Ersetzungsschritten wird auch als Reduktion oder Reduktionskette bezeichnet. x, y, z bezeichnen (ggf. indiziert) stets Variablen. Kleine griechische Buchstaben bezeichnen normalerweise Substitutionen; wenn nicht, wird explizit darauf hingewiesen. s und t sind üblicherweise Terme, o, p und q Positionen.
Abgesehen von auch im Deutschen üblichen Anglizismen wie Heap, Hashtabelle, Choice-Point oder lazy Position werden normalerweise deutsche Wörter verwendet. Eine Ausnahme bilden Positionsbezeichnungen wie leftmost-innermost oder leftmost-outermost. Wenn (aus sprachlichen Gründen) von Gleichungsmengen die Rede ist, sind eigentlich Folgen gemeint, d.h. “Gleichungsmengen” sind geordnete Multimengen. Bei diesen Gleichungsmengen werden die Mengenklammern üblicherweise ausgelassen.

Kapitel 2

Narrowing erster Ordnung

Narrowing ist ein vielen funktional-logischen Sprachen zugrundeliegender Auswertungsmechanismus. Diese Sprachen integrieren die großen Vertreter des deklarativen Paradigmas, nämlich die funktionale und die logische Programmierung. Narrowing ist insofern eine Verallgemeinerung der Termersetzung, als Variablen des zu reduzierenden Terms gebunden werden können ("Unifikation statt Matching").
Ein Termersetzungsschritt $\rightarrow$ bzgl. eines TES $\mathcal{R}$ ist wie folgt definiert ([BA95]):

$$t \rightarrow s \Leftrightarrow \exists p \in \overline{\mathcal{O}}(t) \exists \sigma \in Subst \exists l \rightarrow r \in \mathcal{R}.t|_p = \sigma(l) \wedge s = t[\sigma(r)]_p,$$

wohingegen ein Narrowingschritt $\rightsquigarrow$ bzgl. $\mathcal{R}$ durch

$$t \rightsquigarrow s \Leftrightarrow \exists p \in \overline{\mathcal{O}}(t) \exists \sigma \in Subst \exists l \rightarrow r \in \mathcal{R}.\sigma(t|_p) = \sigma(l) \wedge s = \sigma(t[r]_p),$$

definiert ist. Dabei wird im zweiten Fall eine frische Regelvariante mit $\mathcal{V}ar(l \rightarrow r) \cap \mathcal{V}ar(t) = \emptyset$ verwendet. Sowohl $\rightarrow$ als auch $\rightsquigarrow$ werden im folgenden ggf. mit der Position, der Substitution oder der angewendeten Regel indiziert.

Beispiel 2.0.1. *(Narrowing vs. Reduktion)*
Es sei

$$\mathcal{R} = \left\{ \begin{array}{rcl} append([], L) & \rightarrow & L, \\ append([K|R], L) & \rightarrow & [K|append(R, L)], \\ [1,2,3] = [1,2,3] & \rightarrow & true \end{array} \right\}$$

ein TES unter Verwendung der üblichen Notation für Listen. Dann gibt es die Narrowing-Ableitung

$$\begin{array}{rll} append([1,2], X) = [1,2,3] & \rightsquigarrow & \big[1|append([2], X)\big] = [1,2,3] \\ & \rightsquigarrow & \big[1|[2|append([], X)]\big] = [1,2,3] \\ & \rightsquigarrow & \big[1|[2|X]\big] = [1,2,3] \\ & \rightsquigarrow_{\{X \mapsto [3]\}} & true \end{array}$$

und die Reduktionskette

$$\begin{aligned}
append([1,2],[3]) = [1,2,3] &\rightarrow \big[1|append([2],[3])\big] = [1,2,3] \\
&\rightarrow \big[1|[2|append([],[3])]\big] = [1,2,3] \\
&\rightarrow \big[1|[2|[3]]\big] = [1,2,3] \\
&\rightarrow true
\end{aligned}$$

■

Narrowing ist intensiv untersucht worden, und so gibt es eine Vielzahl von Kalkülen: Simple, Basic, Innermost, Normalizing, Outermost, Lazy, (weakly) Needed, Parallel, Conditional, Outside-In und Applicative Narrowing, um nur einige zu nennen (vgl. [Han94] für einen Überblick).
In dieser Arbeit werden verschiedene Kalküle (OINC [IN97], NCA [NMI95]) mit Needed Narrowing ([AEH94a],[AEH94b]) verglichen, das Eigenschaften wie Optimalität in bezug auf die Länge der Ableitung bei term-sharing, Unvergleichbarkeit von Lösungen und Determinismus auf Grundtermen besitzt.
Im folgenden wird zunächst Needed Narrowing und dann OINC (Outside-In Narrowing Calculus) vorgestellt.

2.1 Needed Narrowing

Die zentrale Datenstruktur von Needed Narrowing (NN) sind die "Definierenden Bäume" ([Ant92]). Definierende Bäume dienen im wesentlichen der Implementierung des Pattern Matching.

Definition 2.1.1. *((partielle) definierende Bäume, [Ant92])*

1. *$\mathcal{T}$ ist ein partieller definierender Baum (pDT) mit Muster $pattern(\mathcal{T}) = \pi$ in bezug auf ein KB-TES $\mathcal{R}$ genau dann, wenn*
   - *$\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, wobei π ein Muster ist, o das Vorkommen einer Variablen in π ist, die Sorte von $\pi|_o$ durch die Konstruktoren $c_1, \ldots, c_k$ für $k > 0$ definiert ist und für alle $1 \leq i \leq k$ $\mathcal{T}_i$ ein pDT mit Muster $\pi[c_i(X_1, \ldots, X_n)]_o$ für $n = Arity(c_i)$ und frische Variablen $X_1, \ldots, X_n$ ist;*
   - *oder $\mathcal{T} = rule(\pi \rightarrow r)$, wobei π ein Muster und $\pi \rightarrow r$ eine Regel aus $\mathcal{R}$ ist;*
   - *oder $\mathcal{T} = exempt(\pi)$ ist, wobei π ein Muster und $l \not\leq \pi$ für alle Regeln $l \rightarrow r$ aus $\mathcal{R}$ ist.*
2. *$\mathcal{T}$ ist ein definierender Baum (DT) einer Funktion f genau dann, wenn $\mathcal{T}$ ein pDT mit Muster $f(X_1, \ldots, X_n)$ für $n = Arity(f)$ und frische Variablen $X_1, \ldots, X_n$ ist.*

△

In [Ant92] wird eine graphische Darstellung von DTs angegeben, auf deren Wiedergabe hier verzichtet wird.

Beispiel 2.1.2. *(definierender Baum)*
Die Funktion $\leq$ mit den definierenden Regeln $\mathcal{R} = \{0 \leq x \rightarrow true, s(x) \leq 0 \rightarrow false, s(x) \leq s(y) \rightarrow x \leq y\}$ besitzt den definierenden Baum

$$
\begin{array}{l}
branch\big(x \leq y,\, 1, \\
\qquad\qquad rule(0 \leq y \rightarrow true), \\
\qquad\qquad branch\big(s(x') \leq y,\, 2, \\
\qquad\qquad\qquad\qquad rule(s(x') \leq 0 \rightarrow false), \\
\qquad\qquad\qquad\qquad rule(s(x') \leq s(y') \rightarrow x' \leq y') \\
\qquad\qquad\qquad \big) \\
\qquad\quad \big)
\end{array}
$$

■

Definierende Bäume sind nicht eindeutig bis auf Isomorphie ([Ant92], Display [8]). Sie können aber automatisch aus der Funktionsdefinition berechnet werden.

Mit DTs kann nun eine Klasse von TES definiert werden, für die NN korrekt und vollständig ist (es gibt eine "Weiterentwicklung" von NN, Parallel Narrowing mit Simplifizierung ([AEH97]), das für konstruktorbasierte fast orthogonale TES korrekt und vollständig (vgl. Abschnitt 3.6), deterministisch auf Grundtermen und für induktiv-sequentielle TES identisch zu NN ist und in diesem Fall die angesprochenen Optimalitätseigenschaften besitzt. Dazu wird eine Erweiterung definierender Bäume benötigt, nämlich generalisierte definierende Bäume).

Definition 2.1.3. *(Induktive Sequentialität, [Ant92])*

1. *Eine in einem TES $\mathcal{R}$ definierte Funktion f heißt induktiv-sequentiell genau dann, wenn es einen DT für f gibt, so daß die enthaltenen Regeln genau die definierenden Regeln für f sind.*

2. *Ein TES $\mathcal{R}$ heißt induktiv sequentiell genau dann, wenn alle in $\mathcal{R}$ definierten Funktionen induktiv-sequentiell sind.*

△

Induktiv-sequentielle TES sind KBO-TES, aber nicht jedes orthogonale oder konstruktorbasierte TES ist induktiv-sequentiell.

Beispiel 2.1.4. *(induktiv-sequentielle TES)*
$\mathcal{R}_1 = \{true \vee x \to true, x \vee true \to true, false \vee false \to false\}$ *ist nicht orthogonal und nicht induktiv-sequentiell.* $\mathcal{R}_2 = \{true \vee x \to true, false \vee x \to x\}$ *ist induktiv-sequentiell.* $\mathcal{R}_3 = \{f(g(a), b) \to a, g(b) \to c\}$ *ist orthogonal, aber nicht induktiv-sequentiell.* $\mathcal{R}_4 = \{f(x, a, b) \to a, f(a, x, c) \to a, f(b, b, x) \to a\}$ *ist orthogonal und konstruktorbasiert, aber nicht induktiv-sequentiell.*

■

Damit kann nun NN definiert werden:

Definition 2.1.5. *(Needed Narrowing, [AEH94a])*
Eine Narrowing-Ableitung ist eine Needed-Narrowing-Ableitung, wenn für jeden Schritt $t \rightsquigarrow_{p,l\to r,\sigma} t'$ *der Ableitung* $(p, l \to r, \sigma) \in \lambda(t, \mathcal{T})$ *ist, wobei* $\mathcal{T}$ *ein DT für* $Head(t)$ *ist und der Bildbereich von* λ *die kleinste Menge mit*

$$\lambda(t,\mathcal{T}) \ni \begin{cases} (\varepsilon, l \to r, mgu(t,l)) & \text{falls } \mathcal{T} = rule(l \to r); \hfill (R_1^\lambda) \\ (\varepsilon, ?, mgu(t,\pi)) & \text{falls } \mathcal{T} = exempt(\pi); \hfill (R_2^\lambda) \\ (p, l \to r, \sigma) & \text{falls } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \exists i:\ t \text{ und } pattern(\mathcal{T}_i) \text{ unifizierbar und} \\ & \quad (p, l \to r, \sigma) \in \lambda(t, \mathcal{T}_i); \hfill (R_3^\lambda) \\ (o \cdot p, l \to r, \sigma \circ \tau) & \text{falls } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \forall i: t \text{ und } pattern(\mathcal{T}_i) \text{ nicht unifizierbar}, \\ & \quad \tau = mgu(t,\pi), \\ & \quad \mathcal{T}' \text{ def. Baum für } \tau(Head(t|_o)) \text{ und} \\ & \quad (p, l \to r, \sigma) \in \lambda(\tau(t|_o), \mathcal{T}'). \hfill (R_4^\lambda) \end{cases}$$

ist.

△

In [AEH94a] werden NN-Ableitungen über den Begriff der outermost-needed Redexe definiert. Eine Position p eines Terms t ist ein (outermost-) needed Redex genau dann, wenn in jeder Reduktion von t zu einer Normalform jeder Nachkomme ("descendant") von $t|_p$ an seiner Wurzelposition reduziert wird. Ein Narrowingschritt $t \rightsquigarrow_{p,l\to r,\sigma} t'$ heißt dann (outermost-) needed genau dann, wenn p für jedes $\vartheta \geq \sigma$ ein (outermost-) needed Redex von $\vartheta(t)$ ist. Eine Narrowingableitung heißt (outermost-) needed genau dann, wenn jeder Schritt der Ableitung (outermost-) needed ist. In [AEH94b] wird nachgewiesen, daß die Funktion λ solche Ableitungen berechnet.

Intuitiv ist das Verhalten von λ für einen Term $t = f(t_1, \ldots, t_n)$, in dem Narrowing durchgeführt werden soll, das folgende: Es wird versucht, t mit einem (nicht unbedingt eindeutig bestimmten) maximal instantiierten Muster π eines Knotens des DT für f zu unifizieren (unter evtl. mehrfacher nichtdeterministischer Anwendung von R_3^λ, was dem Abstieg in *einem* DT entspricht). Es sei $\tau = mgu(t, \pi)$ und $\mathcal{T}$ der pDT, dessen Muster π maximal instantiiert ist. Wenn $\mathcal{T}$ ein *rule*-pDT ist, dann wird ein Narrowingschritt an der Wurzelposition von $\tau(t)$ mit der im pDT enthaltenen Regel durchgeführt (R_1^λ). Wenn $\mathcal{T}$ ein *exempt*-pDT ist, dann gibt es keine Narrowingsequenz, die $\tau(t)$ in Kopfnormalform überführt (R_2^λ). Wenn $\mathcal{T}$ ein *branch*-pDT ist, dann ist $\tau(t)$ mit keinem Muster der Unter-pDTs von $\mathcal{T}$ unifizierbar, weil andernfalls π nicht maximal wäre. Dann wird das Verfahren für $\tau(t)|_o$ (rekursiv) wiederholt, wobei o die Verzweigungsposition des Wurzelknotens des pDT $\mathcal{T}$ ist (R_4^λ). τ ist in diesem Fall eine antizipierte Substitution, was dazu führt, daß NN kein most-general Narrowing ist, d.h. die Substitution eines jeden Schrittes ist nicht unbedingt der mgu des Terms an der Narrowingposition mit einer linken Regelseite.

Beispiel 2.1.6. *(NN)*
$\mathcal{R} = \{0 + x \to x, s(x) + y \to s(x + y), 0 \leq x \to true, s(x) \leq 0 \to false, s(x) \leq s(y) \to x \leq y\}$ *sei ein induktiv-sequentielles TES. Für den Term* $x \leq y + z$ *gibt es u.a. die NN-Ableitungen*

$$x \leq y + z \rightsquigarrow_{\varepsilon, 0 \leq x_1 \to true, \{x \mapsto 0\}} true$$

mit der errechneten Substitution $\{x \mapsto 0\}$ *und*

$$\begin{array}{lll} x \leq y + z & \rightsquigarrow_{1, s(x_2)+y_2 \to s(x_2+y_2), \{x \mapsto s(x_1), y \mapsto s(x_2), z \mapsto y_2\}} & s(x_1) \leq s(x_2 + y_2) \\ & \rightsquigarrow_{\varepsilon, s(x_3) \leq s(y_3) \to x_3 \leq y_3, \{x_1 \mapsto x_3, y_3 \mapsto x_2+y_2\}} & x_3 \leq x_2 + y_2 \\ & \rightsquigarrow_{\varepsilon, 0 \leq x_4 \to true, \{x_3 \mapsto 0, x_4 \mapsto x_2+y_2\}} & true \end{array}$$

mit der errechneten Substitution $\{x \mapsto s(0), y \mapsto s(x_2), z \mapsto y_2\}$.

■

NN berechnet nur "outermost-needed"-Schritte [AEH94a], berechnet (bei termsharing) nur Ableitungen minimaler Länge, ist deterministisch auf Grundtermen, und die errechneten Substitutionen sind unabhängig voneinander, d.h. für zwei errechnete Substitutionen σ und ϑ gibt es ein $x \in \mathcal{V}ar$, so daß $\sigma(x)$ und $\vartheta(x)$ nicht unifizierbar sind. Damit sind die von NN berechneten Substitutionen insbesondere allgemeinste Substitutionen, und niemals wird eine Lösung mehrfach berechnet.
Nichtdeterminismus kann nur bei Auftreten des Falls R_3^λ auftreten. Dann müssen natürlich alle Möglichkeiten ausprobiert werden, d.h. es muß in alle Unterbäume abgestiegen werden, deren Muster mit dem abzuleitenden Term unifizierbar ist.

Korrektheit und Vollständigkeit von NN werden bzgl. der strikten Gleichheit nachgewiesen. Deshalb werden diese Resultate erst nach Definition der strikten Gleichheit in Abschnitt 2.3 zitiert.

2.2 OINC

Der zweite in dieser Arbeit untersuchte Narrowing-Kalkül erster Ordnung ist der "Outside-In Narrowing Calculus" (OINC, [IN97], [IN94]), der NN sehr ähnlich ist, obwohl seine Definition (über ein Inferenzschema) sich grundlegend von der für NN unterscheidet. Das zentrale Resultat von Kapitel 3 wird sein, daß man jede NN-Ableitung durch eine OINC-Ableitung simulieren kann und umgekehrt.

Die Definition von OINC resultiert aus der Untersuchung einer Ordnung über Narrowing-Ableitungen. [IN94] definieren eine Standard-Narrowing-Ableitung in Analogie zu den Standardreduktionen in [HL91] und definieren OINC dann über ein Inferenzschema so, daß dieser Kalkül genau die Standard-Narrowing-Ableitungen berechnet.

Definition 2.2.1. *(OINC, [IN97])*
OINC ist für ein OTES $\mathcal{R}$ über folgendes Inferenzschema definiert:

- *[on], outermost Narrowing*

$$\frac{f(s_1,\ldots,s_n) \approx t, E}{s_1 \approx l_1,\ldots,s_n \approx l_n, r \approx t, E}, t \notin \mathcal{V}ar,$$

  falls es eine frische Regel $f(l_1,\ldots,l_n) \to r \in \mathcal{R}$ gibt;

- *[d], Dekomposition*

$$\frac{f(s_1,\ldots,s_n) \approx f(t_1,\ldots,t_n), E}{s_1 \approx t_1,\ldots,s_n \approx t_n, E}$$

- *[v], Variablenelimination*

$$\frac{t \approx x, E}{\{x \mapsto t\}(E)} \text{ und } \frac{x \approx t, E}{\{x \mapsto t\}(E)}, t \notin \mathcal{V}ar$$

Die Bezeichnungen für die Inferenzregeln werden später ggf. als Index für $\rightsquigarrow$ verwendet.

$\triangle$

OINC rechnet also im Unterschied zu NN mit Gleichungen. Die Semantik von $t \approx s$ ist zunächst nicht formal: "Sind t und s modulo $\mathcal{R}$ unifizierbar?"
Auf diesen Punkt wird weiter unten noch eingegangen werden.
Auffällig ist weiterhin, daß keine Unterscheiduung zwischen Konstruktoren und definierten Funktionsymbolen getroffen wird (vgl. Kapitel 4).
OINC ist nur für eine bestimmte Klasse initialer Gleichungen (Ziele, Anfragen, d.h. Gleichungen, mit denen die Berechnung begonnen wird) korrekt und vollständig (bei OTES). Diese Klasse ist die Klasse der "rechtsnormalen Gleichungen".

Definition 2.2.2. *(rechtsnormal, [IN94])*
Eine Gleichung $s \approx t$ heißt rechtsnormal, falls t in Grundnormalform ist. Eine Gleichunsgmenge G heißt rechtsnormal, wenn jede Gleichung aus G rechtsnormal ist.

$\triangle$

Definition 2.2.3. *(OINC-geeignet, [IN94])*
Eine Gleichung $s \approx t$ heißt OINC-geeignet, falls es eine rechtsnormale Gleichungsmenge G und eine OINC-Ableitung $G \leadsto^* s \approx t, E$ gibt. Eine Gleichungsmenge E heißt OINC-geeignet, falls jedes $s \approx t \in E$ OINC-geeignet ist.

$\triangle$

Die Einschränkung auf rechtsnormale Gleichungen und orthogonale TES erklärt die Abwesenheit einer symmetrischen [on]-Regel.

Auch im folgenden wird aus sprachlichen Gründen stets von "Gleichungsmengen" die Rede sein, obwohl diese natürlich geordnet sind, Elemente mehrfach enthalten können und deshalb eigentlich "Gleichungsfolgen" sind. Deshalb werden auch die Mengenklammern weggelassen.
Interessant sind natürlich die OINC-Ableitungen, die schließlich terminieren und "erfolgreich" sind, also eine "Lösung" berechnen:

Definition 2.2.4. *($\Box$, erfolgreiche Ableitung, [IN94])*
$\Box$ bezeichne die leere Gleichungsmenge. Eine von einer Anfrage G ausgehende OINC-Ableitung $G \leadsto^* \Box$ heißt erfolgreich.

$\triangle$

Beispiel 2.2.5. *(OINC)*
$\mathcal{R} = \{0 + x \to x, s(x) + y \to s(x + y), 0 \leq x \to true, s(x) \leq 0 \to false, s(x) \leq s(y) \to x \leq y\}$ sei das TES aus Beispiel 2.1.6.
Für die Anfrage $x \leq y + z \approx true$ gibt es u.a. die OINC-Sequenzen

$$\begin{array}{lll} x \leq y + z \approx true & \leadsto_{[on],0 \leq x_1 \to true} & x \approx 0, y + z \approx x_1, true \approx true \\ & \leadsto_{[v],\{x \mapsto 0\}} & y + z \approx x_1, true \approx true \\ & \leadsto_{[v],\{x_1 \mapsto y+z\}} & true \approx true \\ & \leadsto_{[d]} & \Box \end{array}$$

mit der errechneten Substitution $\{x \mapsto 0\}$ und

$$\begin{array}{lll}
x \le y + z \approx true & \leadsto_{[on],s(x_1)\le s(y_1)\to x_1\le y_1} & x \approx s(x_1), y+z \approx s(y_1), \\
& & x_1 \le y_1 \approx true \\
& \leadsto_{[v],\{x\mapsto s(x_1)\}} & y+z \approx s(y_1), x_1 \le y_1 \approx true \\
& \leadsto_{[on],s(x_2)+y_2\to s(x_2+y_2)} & y \approx s(x_2), z \approx y_2, \\
& & s(x_2+y_2) \approx s(y_1), x_1 \le y_1 \approx true \\
& \leadsto_{[v],\{y\mapsto s(x_2)\}} & z \approx y_2, s(x_2+y_2) \approx s(y_1), \\
& & x_1 \le y_1 \approx true \\
& \leadsto_{[v],\{y_2\mapsto z\}} & s(x_2+z) \approx s(y_1), x_1 \le y_1 \approx true \\
& \leadsto_{[d]} & x_2+z \approx y_1, x_1 \le y_1 \approx true \\
& \leadsto_{[v],\{y_1\mapsto x_2+z\}} & x_1 \le x_2+z \approx true \\
& \leadsto_{[on],0\le x_3\to true} & x_1 \approx 0, x_2+z \approx x_3, true \approx true \\
& \leadsto_{[v],\{x_1\mapsto 0\}} & x_2+z \approx x_3, true \approx true \\
& \leadsto_{[v],\{x_3\mapsto x_2+z\}} & true \approx true \\
& \leadsto_{[d]} & \Box
\end{array}$$

mit der errechneten Substitution $\{x \mapsto s(0), y \mapsto s(x_2)\}$.

■

In OINC gibt es zwei Quellen für Nichtdeterminismus. Einerseits müssen in einem [on]-Schritt alle Regeln ausprobiert werden, und andererseits gibt es Nichtdeterminismus zwischen [on] und [d]:

Beispiel 2.2.6. *(Nichtdeterminismus bei [on] und [d], vgl. Bsp. 5.1 aus [IN94])*
$\mathcal{R} = \{f(g(d)) \to a, g(c) \to g(d)\}$ *sei ein TES.*
Für die Anfrage $f(g(x)) \approx a$ *gibt es die OINC-Ableitungen*

$$\begin{array}{lll}
f(g(x)) \approx a & \leadsto_{[on]} & g(x) \approx g(d), a \approx a \\
& \leadsto_{[on]} & x \approx c, g(d) \approx g(d), a \approx a \\
& \leadsto_{[v],\{x\mapsto c\}} & g(d) \approx g(d), a \approx a \\
& \leadsto_{[d]} & d \approx d, a \approx a \\
& \leadsto^2_{[d]} & \Box
\end{array}$$

mit der errechneten Substitution $\{x \mapsto c\}$ *und*

$$\begin{array}{lll}
f(g(x)) \approx a & \leadsto_{[on]} & g(x) \approx g(d), a \approx a \\
& \leadsto_{[d]} & x \approx d, a \approx a \\
& \leadsto_{[v],\{x\mapsto d\}} & a \approx a \\
& \leadsto_{[d]} & \Box
\end{array}$$

mit der errechneten Substitution $\{x \mapsto d\}$.

■

Da das Symbol $\approx$ nicht in der Signatur der betrachteten TES $\mathcal{R}$ vorkommt, werden Korrektheit und Vollständigkeit bzgl. $\mathcal{R}_+ = \mathcal{R} \cup \{x \approx x \to true\}$ definiert ($\mathcal{R}_+$ ist nicht orthogonal). Dazu bezeichne $\top$ generisch ein oder mehrere $true$s.
Bisher wurde stets von "errechneten Substitutionen" gesprochen. Da OINC auf Gleichungen rechnet, liegt die Suche nach einem Lösungsbegriff nahe.

Definition 2.2.7. *(Lösung, korrekte Lösung, [IN94])*
$\mathcal{R}$ sei ein OTES und G eine OINC-geeignete Gleichungsmenge.

1. *Eine Substitution $\sigma = \vartheta[\mathcal{V}ar(G)]$ ist eine Lösung von G, falls es eine OINC-Ableitung $G \leadsto^*_\vartheta \square$ gibt.*
2. *Eine Lösung σ von G ist korrekt, falls es eine Reduktion $\sigma(G) \to^*_{\mathcal{R}_+} \top$ gibt.*

$\triangle$

Satz 2.2.8. *(Korrektheit und Vollständigkeit von OINC)*
G sei eine OINC-geeignete Gleichunsgmenge (eine Anfrage) und $\mathcal{R}$ ein orthogonales TES.

1. *(Korrektheit, Prop. 5.1 in [IN94]) Wenn es eine OINC-Ableitung $G \leadsto^*_\vartheta \square$ bzgl. $\mathcal{R}$ gibt, dann gibt es eine Reduktion $\vartheta(G) \to^*_{\mathcal{R}_+} \top$.*
2. *(Vollständigkeit, Theorem 5.1 in [IN94]) Wenn ϑ eine korrekte normalisierbare Lösung von G ist, dann gibt es eine OINC-Ableitung $G \leadsto^*_\sigma \square$ bzgl. $\mathcal{R}$ mit $\sigma \leq_\mathcal{R} \vartheta[\mathcal{V}ar(G)]$. Im Fall konstruktorbasierter TES kann der Index $\mathcal{R}$ weggelassen werden.*

Das Problem der reflexiven Gleichheit liegt darin, daß sie für unendliche Objekte i.a. nicht semientscheidbar ist ([GLMP91]).

2.3 Strikte Gleichheit

Im Kontext der funktional-logischen Programmierung interessiert man sich deshalb für die sog. strikte Gleichheit ([GLMP91], [AEH94a]), um mit evtl. nichtterminierenden TES umgehen zu können (denn in diesen müssen Normalformen nicht existieren).
Strikte Gleichheit zweier Terme ist genau dann gegeben, wenn diese zu demselben Grundkonstruktorterm reduziert werden können. Eine Substitution ϑ ist also Lösung einer strikten Gleichung $s \equiv t$, falls $\vartheta(s)$ und $\vartheta(t)$ zu demselben Grundkonstruktorterm reduzierbar sind.
Man kann die strikte Gleichheit über ein TES definieren:

Definition 2.3.1. *(strikte Gleichheit, $\equiv$, [AEH94b])*
Die strikte Gleichheit $\equiv$ läßt sich über das folgende induktiv-sequentielle TES definieren:

$$\left\{ \begin{array}{rcl} c \equiv c & \to & true \\ c(x_1, \ \ldots, x_n) \equiv c(y_1, \ \ldots, y_n) & \to & \bigwedge_{i=1}^{n}(x_i \equiv y_i) \\ true \wedge x & \to & x, \end{array} \right\}$$

wobei

1. *$c \in \mathcal{C}$ mit $Arity(c) = 0$ in der ersten Regel, $Arity(c) = n$ (> 0) in der zweiten und $\{x_1, \ldots, x_n, y_1, \ldots y_n\} \subseteq \mathcal{V}ar$ ist und*

2. *$\wedge$ ein rechtsassoziatives binäres Funktionssymbol ist.*

$\triangle$

Im folgenden bezeichne $\mathcal{R}^{\equiv}$ das um die Regeln für $\equiv$ und $\wedge$ erweiterte TES $\mathcal{R}$, das $\equiv$ und $\wedge$ nicht in der Signatur enthält.
$\mathcal{R}^{\equiv}$ ist orthogonal, falls $\mathcal{R}$ orthogonal ist. Außerdem gilt

Lemma 2.3.2. *($\equiv$ und Reduktion zu true, Prop. 1 in [AEH94b])*
Es sei $\mathcal{R}$ ein TES, dessen Signatur $\equiv$ und $\wedge$ nicht enthält. Dann sind für alle Terme s und t äquivalent:

1. *s und t sind bzgl. $\mathcal{R}$ zu einem identischen Grundkonstruktorterm reduzierbar.*

2. *$s \equiv t$ ist bzgl. $\mathcal{R}^{\equiv}$ zu true reduzierbar.*

Beispiel 2.3.3. *($\equiv$ (1))*
Für das TES $\mathcal{R} = \{f(a) \mapsto c\}$ und die Anfrage $f(x) \equiv c$ gibt es bzgl. $\mathcal{R}^{\equiv}$ die NN-Ableitung

$$f(x) \equiv c \leadsto_{\{x \mapsto a\}} c \equiv c \leadsto true$$

und die OINC-Ableitung

$$\begin{array}{lll} f(x) \equiv c \approx true & \leadsto_{[on]} & f(x) \approx c, c \approx c, true \approx true \\ & \leadsto_{[on]} & x \approx a, c \approx c, c \approx c, true \approx true \\ & \leadsto_{[v],\{x \mapsto a\}} & c \approx c, c \approx c, true \approx true \\ & \leadsto_{[d]}^{3} & \square \end{array}$$

$\blacksquare$

NN ist nun korrekt und vollständig für induktiv-sequentielle TES bzgl. strikter Gleichungen:

Satz 2.3.4. *(Korrektheit und Vollständigkeit, Theoreme 2 und 4 in [AEH94b])* *$\mathcal{R}$ sei ein induktiv-sequentielles TES.*

1. *(Korrektheit) Wenn $t \equiv t' \rightsquigarrow_\sigma^* true$ eine NN-Ableitung bzgl. $\mathcal{R}^\equiv$ ist, dann gibt es eine Reduktion $\sigma(t) \equiv \sigma(t') \to_{\mathcal{R}^\equiv}^* true$.*
2. *(Vollständigkeit) σ sei eine Konstruktorsubstitution, die Lösung für $t \equiv t'$ ist, d.h. $\sigma(t) \equiv \sigma(t') \to_{\mathcal{R}^\equiv}^* true$. $\mathcal{W}$ sei eine endliche Variablenmenge mit $\mathcal{V}ar(t) \cup \mathcal{V}ar(t') \subseteq \mathcal{W}$. Dann gibt es eine NN-Ableitung $t \equiv t' \rightsquigarrow_{\sigma'}^* true$ mit $\sigma' \leq \sigma[\mathcal{W}]$.*

Die resultierenden Ableitungen können – gerade im Fall von OINC – sehr lang werden:

Beispiel 2.3.5. *($\equiv$ (2))*
Es sei $\mathcal{R} = \{rfirst(0) \to [], rfirst(s(x)) \to [s(x)|rfirst(x)]\}$ ein TES unter Verwendung der üblichen Listennotation. Die Funktion $rfirst(n)$ berechnet die ersten n natürlichen Zahlen in absteigend sortierter Reihenfolge (r steht für "reversed"),
Dann gibt es bzgl. $\mathcal{R}^\equiv$ für die Anfrage $A = rfirst(x) \equiv [s(0)]$ die NN-Sequenz

$$
\begin{array}{rll}
rfirst(x) \equiv [s(0)] & \rightsquigarrow_{\{x \mapsto s(x')\}} & [s(x')|rfirst(x')] \equiv [s(0)] \\
& \rightsquigarrow & s(x') \equiv s(0) \wedge rfirst(x') \equiv [] \\
& \rightsquigarrow & x' \equiv 0 \wedge rfirst(x') \equiv [] \\
& \rightsquigarrow_{\{x' \mapsto 0\}} & true \wedge rfirst(0) \equiv [] \\
& \rightsquigarrow & rfirst(0) \equiv [] \\
& \rightsquigarrow & [] \equiv [] \\
& \rightsquigarrow & true
\end{array}
$$

mit Lösung $\{x \mapsto s(0)\}$ und die OINC-Sequenz

$$
\begin{array}{rll}
A \approx true & \rightsquigarrow_{[on]} & rfirst(x) \approx [x_1|y_1], [s(0)] \approx [x_1'|y_1'], \\
& & x_1 \equiv x_1' \wedge y_1 \equiv y_1' \approx true \\
& \rightsquigarrow_{[on]} & x \approx s(x_2), [s(x_2)|rfirst(x_2)] \approx [x_1|y_1], [s(0)] \approx [x_1'|y_1'], \\
& & x_1 \equiv x_1' \wedge y_1 \equiv y_1' \approx true \\
& \rightsquigarrow_{[v]} & [s(x_2)|rfirst(x_2)] \approx [x_1|y_1], [s(0)] \approx [x_1'|y_1'], \\
& & x_1 \equiv x_1' \wedge y_1 \equiv y_1' \approx true \\
& \rightsquigarrow_{[d]} & s(x_2) \approx x_1, rfirst(x_2) \approx y_1, [s(0)] \approx [x_1'|y_1'], \\
& & x_1 \equiv x_1' \wedge y_1 \equiv y_1' \approx true \\
& \rightsquigarrow_{[v]}^2 & [s(0)] \approx [x_1'|y_1'], s(x_2) \equiv x_1' \wedge rfirst(x_2) \equiv y_1' \approx true
\end{array}
$$

$$
\begin{array}{ll}
\rightsquigarrow^3_{[d,v]} & s(x_2) \equiv s(0) \wedge rfirst(x_2) \equiv [] \approx true \\
\rightsquigarrow_{[on]} & s(x_2) \equiv s(0) \approx true, rfirst(x_2) \equiv [] \approx x_3, x_3 \approx true \\
\rightsquigarrow_{[on]} & s(x_2) \approx s(x_4), s(0) \approx s(y_4), x_4 \equiv y_4 \approx true, \\
 & rfirst(x_2) \equiv [] \approx x_3, x_3 \approx true \\
\rightsquigarrow^4_{[d,v]} & x_2 \equiv 0 \approx true, rfirst(x_2) \equiv [] \approx x_3, x_3 \approx true \\
\rightsquigarrow^4_{[on,v,d]} & rfirst(0) \equiv [] \approx x_3, x_3 \approx true \\
\rightsquigarrow_{[on]} & rfirst(0) \approx [], [] \approx [], true \approx x_3, x_3 \approx true \\
\rightsquigarrow_{[on]} & 0 \approx 0, [] \approx [], [] \approx [], true \approx x_3, x_3 \approx true \\
\rightsquigarrow^5_{[d,v]]} & \square
\end{array}
$$

mit Lösung $\{x \mapsto s(0)\}$.

■

Ableitungen dieser Art sind Motivation für die Definition eines neuen Kalküls, s-OINC.

2.4 s-OINC

OINC wird zu s-OINC modifiziert, indem die Regeln für die strikte Gleichheit in den Kalkül "hineincodiert" werden. Motiviert wird s-OINC über folgendes Beispiel:

Beispiel 2.4.1. *([IN94])*
Es sei $\mathcal{R} = \{f(z) \to z, g(1) \to 1\}$ *ein TES und* $f(g(x)) \equiv f(y) \approx true$ *ein Ziel. Dann gibt es mit Bezug auf* $\mathcal{R}^{\equiv}$ *folgende OINC-Sequenz:*

$$
\begin{array}{lll}
f(g(x)) \equiv f(y) \approx true & \rightsquigarrow_{[on]} & f(g(x)) \approx 1, f(y) \approx 1, true \approx true \\
 & \rightsquigarrow_{[on]} & g(x) \approx z, z \approx 1, f(y) \approx 1, true \approx true \\
 & \rightsquigarrow_{\{z \mapsto g(x)\}} & g(x) \approx 1, f(y) \approx 1, true \approx true \\
 & \rightsquigarrow_{[on]} & x \approx 1, 1 \approx 1, f(y) \approx 1, true \approx true \\
 & \rightsquigarrow^2_{\{x \mapsto 1\}} & f(y) \approx 1, true \approx true \\
 & \rightsquigarrow_{[on]} & y \approx z', z' \approx 1, true \approx true \\
 & \rightsquigarrow^3_{\{z' \mapsto y, y \mapsto 1\}} & \square.
\end{array}
$$

■

Diese Ableitungen sieht redundant und "unnatürlich" lang aus.

Die Idee ist nun, linke und rechte strikte Gleichungsseiten unabhängig voneinander zu reduzieren, und dann die Kopfkonstruktoren zu vergleichen. Das hat den Vorteil, daß die strikten Regeln (von denen es i.a. sehr viele gibt) nicht *vor* der Berechnung der beiden Gleichungsseiten angewendet werden müssen.

Definition 2.4.2. *(s-OINC, [IN94])*
Der Kalkül s-OINC wird über das folgende Inferenzschema definiert:

- *[ons], outermost Narrowing für strikte Gleichungen*

  $$\frac{f(s_1,\ldots,s_n)\equiv t,E}{s_1\approx l_1,\ldots,s_n\approx l_n,r\equiv t,E}\text{ und }\frac{s\equiv f(t_1\ldots,t_n),E}{t_1\approx l_1,\ldots,t_n\approx l_n,s\equiv r,E}$$

  für eine frische Regel $f(l_1,\ldots,l_n)\to r$

- *[ds], Dekomposition strikter Gleichungen*

  $$\frac{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n),E}{s_1\equiv t_1,\ldots,s_n\equiv t_n,E}$$

- *[ims], Imitierung für strikte Gleichungen*

  $$\frac{c(s_1,\ldots,s_n)\equiv y,E}{\vartheta(s_1\equiv y_1,\ldots s_n\equiv y_n,E)}\text{ und }\frac{y\equiv c(t_1,\ldots,t_n),E}{\vartheta(y_1\equiv t_1,\ldots y_n\equiv t_n,E)}$$

  für $c\in\mathcal{C}$ *und* $\vartheta=\{y\mapsto c(y_1,\ldots,y_n)\}$

- *[ts], Elimination trivialer strikter Gleichungen*

  $$\frac{x\equiv y,E}{\sigma(E)}$$

  $$mit\ \sigma=\begin{cases}\{x\mapsto y\} & falls\ x\neq y(syntaktische\ Gleichheit)\\ \emptyset & sonst\end{cases}$$

- *und den Regeln [on], [d] und [v] aus OINC.*

△

Beispiel 2.4.3. *(Fortsetzung von Bsp. 2.4.1*
Für das TES aus Beispiel 2.4.1 ergibt sich die s-OINC-Sequenz

$$\begin{array}{rll} f(g(x))\equiv f(y) & \rightsquigarrow_{[ons]} & g(x)\approx z,z\equiv f(y)\\ & \rightsquigarrow_{[on]} & x\approx 1,1\approx z,z\equiv f(y)\\ & \rightsquigarrow^2_{\{x\mapsto 1,z\mapsto 1\}} & 1\equiv f(y)\\ & \rightsquigarrow_{[ons]} & y\approx z',1\equiv z'\\ & \rightsquigarrow_{\{z'\mapsto y\}} & 1\equiv y\\ & \rightsquigarrow_{[ims],\{y\mapsto 1\}} & \Box. \end{array}$$

mit der Lösung (eingeschränkt auf die Variablen aus der initialen Gleichung) $\{x\mapsto 1,y\mapsto 1\}$.

■

Beispiel 2.4.4. *(Fortsetzung von Bsp. 2.3.5)*
Es sei $\mathcal{R}$ das TES aus Bsp. 2.3.5. Für die Anfrage $rfirst(x) \equiv [s(0)]$ gibt es die s-OINC-Ableitung

$$\begin{array}{lll} rfirst(x) \equiv [s(0)] & \rightsquigarrow_{[ons]} & x \approx s(x_1), [s(x_1)|rfirst(x_1)] \equiv [s(0)] \\ & \rightsquigarrow_{[v]} & [s(x)|rfirst(x)] \equiv [s(0)] \\ & \rightsquigarrow_{[ds]} & s(x) \equiv s(0), rfirst(x) \equiv [] \\ & \rightsquigarrow_{[ds,ims]} & rfirst(0) \equiv [] \\ & \rightsquigarrow_{[ons]} & 0 \approx 0, [] \equiv [] \\ & \rightsquigarrow_{[d,ds]} & \Box \end{array}$$

■

Der Begriff der OINC-geeigneten Gleichungsmenge wird direkt auf den Begriff der s-OINC-geeigneten Gleichungsmenge übertragen. Bevor die Korrektheits- und Vollständigkeitsresultate zitiert werden, soll auf drei Besonderheiten aufmerksam gemacht werden:

Bemerkung 2.4.5.

1. *Es besteht Nichtdeterminismus zwischen den beiden [ons]-Regeln. Wenn $s \equiv t$ mit $Head(s) \in \mathcal{F}$ und $Head(t) \in \mathcal{F}$ eine zu lösende Gleichung ist, dann werden beide Regeln angewendet. Damit werden natürlich auch identische Lösungen mehrfach berechnet. In Lemma 3.5.3 wird gezeigt, daß man sich in induktiv-sequentiellen TES auf die Auswahl einer Inferenzregel beschränken kann.*

2. *Die Inferenzregel [ts] ermöglicht die Berechnung allgemeinerer Lösungen, als NN und OINC berechnen können.*

3. *Die s-OINC zugrundeliegenden TES enthalten die Regeln für die strikte Gleichheit nicht. Damit kann sie insbesondere nicht auf rechten Gleichungsseiten verwendet werden, was beispielsweise für Implementierungen von $if-then-else$ ungünstige Konsequenzen hat (wenn sie nicht explizit modelliert wird; vgl. auch Bsp. 4.7.3). Eine Implementierung der Fakultätsfunktion durch $fac(x) \rightarrow ite(x \equiv 0, 1, mult(x, pred(x)))$ mit entsprechenden Regeln für das $if-then-else$-Konstrukt ite, die Multiplikation mult und die Vorgängerfunktion pred zeigt die Problematik sofort, wenn $x \equiv 0$ mit dem ersten Argument einer Regel für ite, z.B. $ite(true, s_1, s_2) \rightarrow s_1$, unifiziert werden soll. Dann entsteht nämlich die Gleichung $x \equiv 0 \approx true$.*

Beispiel 2.4.6. *(doppelte Lösungen)*
Für das TES aus Beispiel 2.4.1 gibt es neben der Ableitung aus Beispiel 2.4.3 außerdem die Ableitung

$$\begin{array}{lll} f(g(x)) \equiv f(y) & \rightsquigarrow_{[ons]} & y \approx z, f(g(x)) \equiv z \\ & \rightsquigarrow_{\{z \mapsto y\}} & f(g(x)) \equiv y \\ & \rightsquigarrow_{[ons]} & g(x) \approx z', z' \equiv y \\ & \rightsquigarrow_{\{z' \mapsto g(x)\}} & g(x) \equiv y \\ & \rightsquigarrow_{[ons]} & x \approx 1, 1 \equiv y \\ & \rightsquigarrow_{\{x \mapsto 1\}} & 1 \equiv y \\ & \rightsquigarrow_{[ims],\{y \mapsto 1\}} & \square. \end{array}$$

mit derselben Lösung (eingeschränkt auf die Variablen aus der initialen Gleichung) $\{x \mapsto 1, y \mapsto 1\}$.

■

Beispiel 2.4.7. *(allgemeinere Lösungen)*
Für das TES aus Beispiel 2.4.1 gibt es mit einem zusätzlichen Konstruktor c der Stelligkeit 1 *die s-OINC-Ableitung*

$$c(f(x)) \equiv y \rightsquigarrow_{[ims]} f(x) \equiv y_1 \rightsquigarrow_{[ons]} x \approx z, z \equiv y_1 \rightsquigarrow_{[v]} x \equiv y_1 \rightsquigarrow_{[ts]} \square$$

mit der errechneten Lösung $\{y \mapsto c(y_1), x \mapsto y_1\}$.

■

Mit Bezug auf $\mathcal{R}^{\equiv}$ ist diese Lösung nicht korrekt, aber jede Grundinstanz dieser Lösung. Dieses Vorgehen ist allerdings praktisch gangbar, weil x und y_1 sonst an alle Konstanten gebunden werden müssen (d.h. an alle Elemente des Herbranduniversums).
Wenn später s-OINC und NN direkt verglichen werden, dann wird davon ausgegangen, daß beide Kalküle die Variablen auf beiden Gleichungsseiten an alle Elemente des Herbranduniversums binden.
In der Praxis wird man natürlich auch für NN eine solche strikte Variablenbindung implementieren.

Satz 2.4.8. *([IN94])*

1. *(Korrektheit) Jede Grundinstanz einer von s-OINC berechneten Lösung für eine s-OINC-geeignete Gleichungsmenge G bzgl. eines OTES* $\mathcal{R}$ *ist Lösung von G bzgl.* $\mathcal{R}^{\equiv}$.

2. *(Vollständigkeit) Für jede normalisierbare Lösung* σ *einer s-OINC-geeigneten Gleichungsmenge G bzgl. eines OTES* $\mathcal{R}$ *berechnet s-OINC eine Lösung* ϑ *mit* $\vartheta \leq_{\mathcal{R}} \sigma[\mathcal{V}ar(G)]$. *Im Fall konstruktorbasierter OTES kann bei Ersetzen der Inferenzregel [ts] der Subskript* $\mathcal{R}$ *weggelassen werden.*

2.5 Diskussion

In diesem Kapitel wurden die für diese Arbeit zentralen Kalküle NN und OINC vorgestellt. Die NN zugrundeliegende zentrale Datenstruktur sind die definierenden Bäume, mit denen Pattern Matching bzw. Reduktion effizient implementiert werden kann. NN führt das Narrowing in Termen durch. Da Narrowing als Gleichungslöser angesehen werden kann, stellte sich die Frage nach einem geeigneten Gleichheitsbegriff. Die reflexive Gleichheit ist für unendliche Objekte i.a. unentscheidbar, woraus die Notwendigkeit eines anderen Gleichheitsbegriff für funktional-logische Sprachen abgeleitet wurde. Diese neue Gleichheit ist die strikte Gleichheit, die genau dann gilt, wenn zwei Terme zu demselben Grundkonstruktorterm abgeleitet werden können.
NN ist korrekt und vollständig bzgl. induktiv-sequentieller TES mit strikter Gleichheit und Konstruktorsubstitutionen. OINC ist korrekt und vollständig bzgl. orthogonaler TES und normalisierbarer Substitutionen.
OINC wurde zu s-OINC erweitert, indem die Regeln für die strikte Gleichheit in den Kalkül "hineincodiert" wurden. Korrektheit und Vollständigkeit gelten wie für OINC.

Der Unterschied zwischen beiden Kalkülen liegt neben den betrachteten unterschiedlichen TES in der Definition des Kalküls: NN ist als Funktion definiert und rechnet mit Termen, wobei die strikte Gleichheit über Terme repräsentiert werden kann. OINC hingegen ist über ein Inferenzschema definiert und rechnet auf Gleichungen. Daraus ergibt sich die Schwierigkeit eines Vergleichs beider Verfahren.

Kapitel 3

Systematischer Vergleich von OINC und NN

In diesem Kapitel wird gezeigt, daß eine gegenseitige Simulationsbeziehung zwischen OINC und NN besteht. Diese Beziehung ist dadurch gekennzeichnet, daß man jeder OINC-Ableitung genau eine NN-Ableitung zuordnen kann und umgekehrt. Es wird gezeigt, wie diese Ableitungen aussehen; die Zuordnung ist also konstruktiv.
Damit sind OINC und NN insofern "äquivalent", als sie dieselben Resultate als Lösungen einer Gleichungsmenge liefern. Es wird gezeigt, daß die verwendeten Regeln in beiden Ableitungen dieselben sind, wenn auch die Reihenfolge unterschiedlich ist, wenn man einen NN-Schritt mit einer [on]-Inferenz in Bezug setzt.

Zuerst wird eine für die folgenden Beweise notwendige Konvention für die definierenden Bäume festgelegt (Kapitel 3.1) Dann wird in Kapitel 3.2 zunächst gezeigt, wie ein NN-Schritt mit einer OINC-Sequenz zusammenhängt. Damit kann dann eine NN-Ableitung durch eine OINC-Ableitung simuliert werden. Eindeutigkeitseigenschaften beider Kalküle führen dann zum Beweis der Simulationsbeziehung zwischen erfolgreichen OINC- und NN-Ableitungen. (Kapitel 3.3). Der Unterschied zwischen beiden Kalkülen liegt (neben der Mächtigkeit der zugrundeliegenden TES) im Rechenverhalten. Dieses wird in Kapitel 3.4 diskutiert und anhand experimenteller Resultate verdeutlicht. Kapitel 3.5 stellt s-OINC und NN gegenüber; wiederum ist eine 1:1-Zuordnung der Ableitungen möglich. Auch hier wird das Rechenverhalten diskutiert und anhand experimenteller Resultate verdeutlicht. Schließlich werden in Kapitel 3.6 die zentralen Resultate dieses Abschnitts wiederholt und die verwendeten Methoden im Vergleich mit anderen diskutiert.

Da im NN-Kalkül kein Pendant zu der Unifikationsgleichheit modulo einem TES ($\approx$) existiert, müssen die zu untersuchenden Gleichungsmengen für OINC eingeschränkt werden.
In NN wird das Narrowing für *Terme* durchgeführt, in OINC für *Gleichungen*. Der Grund für die Einschränkung in NN ist die Zielsetzung, Berechnungen für

funktional-logische Programme durchzuführen. Dort interessiert man sich für die strikte Gleichheit, die als Funktion definiert werden kann (s. Kapitel 2). Wenn man nun TES mit strikter Gleichheit betrachtet, so kann davon ausgegangen werden, daß die rechten Seiten der initialen (OINC-)Gleichungen $s \approx t$ i.a. $true$ oder $false$ sind. Diese Gleichungen sind rechtsnormal.
Im folgenden werden als "Startgleichungen" für OINC stets OINC-geeignete Gleichungen gefordert. Das ist angesichts der Tatsache, daß die initialen Gleichungen als rechte Seiten $true$ oder $false$ besitzen und auf der linken Seite beliebige Terme stehen können, keine Einschränkung (abgesehen davon, daß andernfalls gar keine OINC-Berechnungen durchgeführt werden könnten).

3.1 Motivation: LR-definierende Bäume

Ein NN-Schritt kann nur in einem solchen Term t durchgeführt werden, für den $Head(t) \in \mathcal{F}$ gilt (vgl. Kap. 7 in [AEH94a]). Diese Annahme wird im folgenden immer implizit vorausgesetzt werden (bei den Termen, die von Interesse sind, ist das stets das Symbol $\equiv$).

Bevor nun grundlegende Überlegungen zum Vergleich beider Kalküle angestellt werden, sollen zwei Ableitungen exemplarisch betrachtet werden.

Beispiel 3.1.1. *(NN vs. OINC)*
Es sei $\mathcal{R} = \{f(x,a) \to b, f(b,b) \to c, g(x,a) \to a, g(x,b) \to b\}$ *ein OTES.*
Für einen auszuwertenden Ausdruck $A = f(g(a, f(a,a)), g(a, f(a,a)))$ *gibt es die (eindeutige) NN-Ableitung*

$$\begin{array}{lll} A & \rightsquigarrow_{2\cdot 2} & f(g(a,f(a,a)),g(a,b)) \\ & \rightsquigarrow_{2} & f(g(a,f(a,a)),b) \\ & \rightsquigarrow_{1\cdot 2} & f(g(a,b),b) \\ & \rightsquigarrow_{1} & f(b,b) \\ & \rightsquigarrow_{\varepsilon} & c \end{array}$$

und die OINC-Ableitung (mit true als rechter Gleichungsseite, s.o.)

$$\begin{array}{lll} A \approx true & \rightsquigarrow_{[on],\varepsilon} & g(a,f(a,a)) \approx b, g(a,f(a,a)) \approx b, c \approx true \\ & \rightsquigarrow_{[on],1} & a \approx x_1, f(a,a) \approx b, b \approx b, g(a,f(a,a)) \approx b, c \approx true \\ & \rightsquigarrow_{[v]} & f(a,a) \approx b, b \approx b, g(a,f(a,a)) \approx b, c \approx true \\ & \rightsquigarrow_{[on],1\cdot 2} & a \approx x_2, a \approx a, b \approx b, b \approx b, g(a,f(a,a)) \approx b, c \approx true \\ & \rightsquigarrow^{4} & g(a,f(a,a)) \approx b, c \approx true \\ & \rightsquigarrow_{[on],2} & a \approx x_3, f(a,a) \approx b, b \approx b, c \approx true \\ & \rightsquigarrow_{[v]} & f(a,a) \approx b, b \approx b, c \approx true \\ & \rightsquigarrow_{[on],2\cdot 2} & a \approx x_4, a \approx a, b \approx b, b \approx b, c \approx true \\ & \rightsquigarrow^{4} & c \approx true. \end{array}$$

Dabei bezeichnen im ersten Fall die $\rightsquigarrow$*-Subskripte und im zweiten die hinteren* $\rightsquigarrow$*-Subskripte die Positionen, an denen ein Narrowing-Schritt durchgeführt wird.*
In NN sind das die Positionen $2 \cdot 2, 2, 1 \cdot 2, 1, \varepsilon$ *und in OINC* $\varepsilon, 1, 1 \cdot 2, 2, 2 \cdot 2$.
■

Die Narrowingpositionen in OINC sind leftmost-outside-in-geordnet, die Positionen in NN rightmost-inside-out. Für OINC ist die angegebene Ordnung der Positionen für alle Ableitungen und alle initialen Ausdrücke gleich, was für NN nicht der Fall ist (Beispiele lassen sich leicht konstruieren, sind aber umfangreich und damit unübersichtlich). Im allgemeinen läßt sich keine offensichtliche allgemeingültige Gesetzmäßigkeit der Narrowingpositionen in NN ausmachen (später wird eine Gesetzmäßigkeit beschrieben, wenn an die definierenden Bäume eine Bedingung gestellt wurde).
Im Beispiel sind die Narrowingpositionen unabhängig von der "vertikalen" Position von rechts nach links geordnet. Beispiele, in denen Narrowing einmal von rechts nach links und einmal von links nach rechts durchgeführt werden, liegen auf der Hand (sind aber umfangreich, wie bereits angemerkt, und werden deshalb hier ausgelassen).
Es gibt aber eine Möglichkeit, das Narrowing von links nach rechts zu erzwingen: Man permutiert die Argumente der Regeln, so daß die Instantiierung der Argumente von links nach rechts erfolgt (damit ergibt sich kein "Semantikverlust", da es sich um eine rein syntaktische Transformation handelt). Außerdem wird gefordert, daß diese Instantiierung auch von links nach rechts durchgeführt wird, falls andere Möglichkeiten bestehen (z.B. in der Funktionsdefinition $f(a, b) \rightarrow a$, bei der sowohl von links nach rechts, als auch von rechts nach links instantiiert werden kann).
Die Motivation für diese Vorgehen liegt darin begründet, daß die *Kalküle* OINC und NN schwierig zu modifizieren sind, ohne den Verlust bekannter Eigenschaften bzw. den erneut zu führenden Nachweis derselben in Kauf zu nehmen, wohingegen die Modifikation verwendeter *Datenstrukturen* nicht unbedingt einen solchen Verlust implizieren muß. In OINC gibt es abgesehen von den Gleichungen bzw. Gleichungsfolgen keine offensichtliche Datenstruktur; in NN sind dies genau die definierenden Bäume.
Im folgenden kommen deshalb permutierte Bäume zum Einsatz, bei denen die Argumente von links nach rechts instantiiert werden. Diese werden als "Links-rechts definierende Bäume" bezeichnet:

Definition 3.1.2. *(Links-rechts definierender Baum, LR-DT)*
Ein Links-rechts definierender Baum (LR-DT) ist ein definierender Baum, in dem alle Argumentpositionen der Muster von links nach rechts instantiiert werden, d.h. für jeden Knoten $v = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ *mit den Nachfolgerknoten* $v_s = branch(\pi', o', \mathcal{T}'_1, \ldots, \mathcal{T}'_{n'})$ *gilt:* $o \prec o'$ *oder* $o \in Left(o')$ *für alle* v_s.
△

Zur Verdeutlichung ein Beispiel:

Beispiel 3.1.3. *(Fortsetzung von Bsp. 3.1.1)*
$\mathcal{R}'$ entstehe aus $\mathcal{R}$ in Bsp. 3.1.1, indem die Argumente der beiden Regeln vertauscht werden, d.h. $\mathcal{R}' = \{f'(a,x) \to b, f'(b,b) \to c, g'(a,x) \to a, g'(b,x) \to b\}$.
A' entstehe analog aus A, d.h. $A' = f'(g'(f'(a,a),a), g'(f'(a,a),a))$.
Dann gibt es die (eindeutige) NN-Ableitung

$$\begin{array}{rll} A' & \rightsquigarrow_{1\cdot 1} & f'(g'(b,a), g'(f'(a,a),a)) \\ & \rightsquigarrow_{1} & f'(b, g'(f'(a,a),a)) \\ & \rightsquigarrow_{2\cdot 1} & f'(b, g'(b,a)) \\ & \rightsquigarrow_{2} & f'(b,b) \\ & \rightsquigarrow_{\varepsilon} & c \end{array}$$

und die OINC-Ableitung

$$\begin{array}{rll} A' \approx true & \rightsquigarrow_{[on],\varepsilon} & g'(f'(a,a),a) \approx b, g'(f'(a,a),a) \approx b, c \approx true \\ & \rightsquigarrow_{[on],1} & f'(a,a) \approx b, a \approx x_1, b \approx b, g'(f'(a,a),a) \approx b, c \approx true \\ & \rightsquigarrow_{[on],1\cdot 1} & a \approx a, a \approx x_2, b \approx b, a \approx x_1, b \approx b, g'(f'(a,a),a) \approx b, c \approx true \\ & \rightsquigarrow^{5} & g'(f'(a,a),a) \approx b, c \approx true \\ & \rightsquigarrow_{[on],2} & f'(a,a) \approx b, a \approx x_3, b \approx b, c \approx true \\ & \rightsquigarrow_{[on],2\cdot 2} & a \approx a, a \approx x_4, b \approx b, a \approx x_3, b \approx b, c \approx true \\ & \rightsquigarrow^{5} & c \approx true \end{array}$$

Wiederum sind die von OINC berechneten Narrowingpositionen leftmost-outside-in geordnet. Für die NN-Positionen wurde nun erreicht, daß die Positionen (unabhängig von der vertikalen Position) von links nach rechts geordnet sind.

■

Die Einschränkung auf LR-DTs ist keine wirkliche: Wenn ein DT existiert, der kein LR-DT ist, so kann ein solcher immer durch Permutation der Argumente berechnet werden (das kann zur Compilezeit geschehen; die Zeitkomplexität ist pro Baum mit einem brutalen Ansatz schlimmstenfalls nichtdeterministisch polynomiell ungefähr in der Anzahl der Positionen der betrachteten linken Regelseite; insbesondere ist ein (exponentielles) Versuch-und-Irrtum-Verfahren ein Entscheidungsverfahren).
Die erhaltenen Resultate für LR-DTs werden nicht mehr explizit verallgemeinert. Wenn im folgenden also ausschließlich von LR-DTs die Rede ist, dann vergegenwärtige man sich die eben getroffene Feststellung.

Im folgenden wird nun zunächst die Simulierbarkeit eines NN-Schrittes durch eine OINC-Sequenz aufgezeigt; danach wird durch Kombination einzelner Schritte bewiesen, daß es für jede erfolgreiche NN-Ableitung eine entsprechende OINC-Ableitung mit demselben Ergebnis gibt.

3.2 Vergleich einzelner NN- und OINC-Schritte

Vor Angabe der folgenden Sachverhalte sei darauf hingewiesen, daß die Gleichheit von Substitutionen ($\sigma = \vartheta$) immer als $\sigma \leq \vartheta \wedge \vartheta \leq \sigma$ aufzufassen ist, daß also insbesondere von Variablenumbenennungen abstrahiert wird (Kap. 1.2).

Ein erfolgreicher NN-Schritt kann als Folge von Aufrufen der Funktion λ aufgefaßt werden, die mit einem Auftreten des Falls R_1^λ abgeschlossen wird (wenn der Schritt nicht erfolgreich ist, endet diese Folge mit einem Fall R_2^λ).
Im folgenden bezeichne nun Σ eine solche Folge von Aufrufen, d.h.

$$\Sigma = I\widetilde{\Sigma}_1\widetilde{\Sigma}_2 \ldots \widetilde{\Sigma}_n,$$

wobei I der initiale Aufruf von λ ist, $\widetilde{\Sigma}_i \in \{R_3^\lambda, R_4^\lambda\}$ für $i < n$ und $\widetilde{\Sigma}_n \in \{R_1^\lambda, R_2^\lambda\}$. Dabei repräsentieren im folgenden Σ_3 beliebig lange Folgen von Auftreten des Falls R_3^λ. Wenn von "erreichten Knoten" die Rede ist, so sind damit stets die Wurzelknoten desjenigen pDT gemeint, der das zweite Argument des letzten λ-Aufrufs darstellte.

Es wird nun zunächst der Zusammenhang von Σ_3 und OINC-Sequenzen festgestellt. Dann folgt der Zusammenhang von Auftreten des Falls R_4^λ mit [on]-Inferenzen, und schließlich werden die einzelnen Folgen zu der Gesamtfolge Σ zusammengesetzt, die wiederum durch eine entsprechende OINC-Sequenz simuliert werden kann.

Dazu wird eine wichtige Eigenschaft des OINC-Kalküls benötigt, nämlich die Tatsache, daß die Inferenzregeln [d] und [v] einen Algorithmus für die Unifikation von Termen darstellen:

Lemma 3.2.1.
Es sei $t \approx s, E$ eine OINC-geeignete Gleichungsmenge. Wenn t und s unifizierbar mit mgu σ sind, dann gibt es eine OINC-Sequenz $t \approx s, E \rightsquigarrow^*_{[d,v]} \sigma'(E)$ mit $\sigma = \sigma'$.

Beweis
siehe z.B. [MM82]. Da die Variablenmengen disjunkt und die Regeln linkslinear sind, kann auf den Occur-Check verzichtet werden.

□

Beispiel 3.2.2. *([d] und [v] berechnen mgu)*
Der mgu von $c(x, c(y, b))$ und $c(a, c(b, z))$ mit $\{x, y, z\} \subseteq \mathcal{V}ar$ wird durch folgende OINC-Sequenz berechnet:

$$
\begin{array}{rll}
c(x, c(y, b)) \approx c(a, c(b, z)) & \rightsquigarrow_{[d]} & x \approx a, c(y, b) \approx c(b, z) \\
& \rightsquigarrow_{[v],\{x \mapsto a\}} & c(y, b) \approx c(b, z) \\
& \rightsquigarrow_{[d]} & y \approx b, b \approx z \\
& \rightsquigarrow_{[v],\{y \mapsto b\}} & b \approx z \\
& \rightsquigarrow_{[v],\{z \mapsto b\}} & \square
\end{array}
$$

mit der resultierenden Substitution $\{x \mapsto a, y \mapsto b, z \mapsto b\}$.

■

Es wird nun gezeigt, daß dem Erreichen eines LR-DT-Knotens mit der Position p eine OINC-Sequenz entspricht, die zu einer Gleichungsmenge führt, an deren erster Stelle der Unterterm des ursprünglichen Terms an p steht (Lemma 3.2.8). Dazu wird zunächst eine Beobachtung des Verhaltens von OINC durchgeführt und danach eine Eigenschaft von LR-DTs aufgezeigt.

Das folgende Lemma resultiert aus dem oben angesprochenen leftmost-outside-in-Charakter von OINC:

Lemma 3.2.3.
Es sei $t \approx s, E$ *eine OINC-geeignete Gleichungsmenge mit* $s \notin \mathcal{V}ar$, $l \to r$ *eine Regel für* t, *d.h.* $Head(l) = Head(t)$, *und* $o \in \mathcal{O}(t) \cap \mathcal{O}(l)$.
Falls

1. $Head(t|_q) = Head(l|_q)$ *für alle* $q \in \mathcal{O}(t) \cap \mathcal{O}(l)$ *mit* $q \prec o$ *und*
2. $t|_q$ *und* $l|_q$ *unifizierbar mit mgu* σ *für alle* $q \in Left(o) \cap \mathcal{O}(l)$ *sind,*

dann gibt es eine OINC-Sequenz

$$t \approx s, E \rightsquigarrow_{[on]} \rightsquigarrow^*_{[d,v]} \sigma(t)|_o \approx \sigma(l)|_o, \sigma(E'), \sigma(E).$$

Der von den Bedingungen (1) und (2) definierte Sachverhalt wird zur formalen Vereinfachung durch eine Funktion $leftmgu^p$ beschrieben, die den "Unifikator zweier Terme links von oder über p" einer Position p berechnet. Dazu werden die Terme rechts über p "abgeschnitten" und der Unifikator der Restterme berechnet:

Definition 3.2.4. $(leftmgu^p(s,t), rightmgu^p(s,t))$
Die Funktion $leftmgu^p(s,t)$ *ist wie folgt definiert:*
$leftmgu^p(s,t) := mgu(s',t')$, *wobei für* $p = p_1 \cdots p_n$ *und für alle* $m < n$ *der Term* s' *(und* t' *analog) wie folgt definiert ist:*
$s'|_{p_1 \cdots p_m \cdot i} = s|_{p_1 \cdots p_m \cdot i}$ *für* $i < p_{m+1}$,
$s'|_{p_1 \cdots p_{m+1}} = Head(s|_{p_1 \cdots p_{m+1}})$ *und*

$s'|_{p_1\cdots p_m\cdot i} = c_{dummy}$ für $Arity(Head(s|_{p_1\cdots p_m})) \geq i > p_{m+1}$ und ein Dummy-Symbol c_{dummy}, das nicht in der Signatur vorkommt.
$rightmgu^p(s,t)$ ist entsprechend für die Positionen rechts von oder über p definiert.

△

Beispiel 3.2.5. *($leftmgu$, $rightmgu$)*
Der Unifikator links von oder über $1 \cdot 2 \cdot 2$, $leftmgu^{1\cdot 2\cdot 2}(t,s)$, von

$$t := f(g(x, h(y,a)), z, c) \text{ und } s := f(g(h(a,b), h(b,a)), a, w)$$

mit $\{w,x,y,z\} \subseteq \mathcal{V}ar$ ist $\{x \mapsto h(a,b), y \mapsto b\}$. Für die Positionen rechts von oder über $1 \cdot 2 \cdot 2$ ist $rightmgu^{1\cdot 2\cdot 2}(t,s) = \{z \mapsto a, w \mapsto c\}$, und es ist $mgu(t,s) = \{x \mapsto h(a,b), y \mapsto b, z \mapsto a, w \mapsto c\}$.

■

Beweis von Lemma 3.2.3
durch Induktion über o.
$o = \varepsilon$ ist trivial.
Induktionsverankerung: $o \neq \varepsilon$ atomare Position:
Es sei $t = f(t_1, \ldots, t_n)$ und $f(l_1, \ldots, l_n) \to r$ eine Regel für f. Dann gibt es die OINC-Sequenz

$$t \approx s, E \leadsto_{[on]} t_1 \approx l_1, \ldots, t_n \approx l_n, r \approx s, E.$$

Es sei $t_j = t|_o$. Nach Voraussetzung sind t und $f(l_1, \ldots, l_n)$ für alle Positionen links oder über p unifizierbar. Mit Lemma 3.2.1 gibt es demnach die Sequenz

$$\begin{aligned} t_1 \approx l_1, \ldots, t_n \approx l_n, r \approx s, E \quad \leadsto^*_{[d,v]} \quad & \sigma(t_j) \approx \sigma(l_j), \ldots, \sigma(t_n) \approx \sigma(l_n), \\ & \sigma(r) \approx \sigma(s), \sigma(E) \end{aligned}$$

mit $\sigma = leftmgu^o\big(t, f(l_1, \ldots, l_n)\big)$.

Induktionsschritt: $o = o' \cdot p$:
Nach Voraussetzung sind die Bedingungen (1) und (2) für o und damit insbesondere für o' erfüllt; demnach gibt es nach Induktionsannahme für eine linke Regelseite l eine OINC-Sequenz

$$t \approx s, E \leadsto^* \sigma(t)|_{o'} \approx \sigma(l)|_{o'}, \sigma(E'), \sigma(E).$$

Wenn $\sigma(l)|_{o'} \in \mathcal{V}ar$, dann ist $o' \cdot p \notin \mathcal{O}(l)$. Wenn $\sigma(t)|_{o'} \in \mathcal{V}ar$, dann ist $o' \cdot p \notin \mathcal{O}(t)$.
Also ist wegen Bedingung (1) $Head(\sigma(t)|_{o'}) = Head(\sigma(l)|_{o'})$ für $\sigma(t)|_{o'} = g(t_1, \ldots, t_n)$ und $\sigma(l)|_{o'} = g(l_1, \ldots, l_n)$, und demnach ist ein [d]-Schritt möglich:

$$\sigma(t)|_{o'} \approx \sigma(l)|_{o'}, \sigma(E'), \sigma(E) \leadsto_{[d]} t_1 \approx l_1, \ldots, t_n \approx l_n, \sigma(E'), \sigma(E).$$

Bedingung (2) garantiert, daß $\sigma(t)|_q$ und $\sigma(l)|_q$ für alle $q \in Left(o)$ mit σ' unifizierbar sind, also insbesondere die Terme t_i und l_i für $i < j$ mit $\sigma(t)|_{o'\cdot p} = t_j$. $j-1$-malige Anwendung von Lemma 3.2.1 liefert die OINC-Sequenz

$$\begin{aligned} t_1 \approx l_1, \dots, t_n \approx l_n, \sigma(E'), \sigma(E) \quad \rightsquigarrow^*_{[d,v]} \quad & \sigma'(t_j) \approx \sigma'(l_j), \dots, \sigma'(t_n) \approx \sigma'(l_n), \\ & \sigma' \circ \sigma(E'), \sigma' \circ \sigma(E). \end{aligned}$$

Wegen $t_j = \sigma(t)|_{o'\cdot p}$ und $l_j = \sigma(l)|_{o'\cdot p}$ ist die letzte Gleichungsmenge äquivalent zu

$$\sigma' \circ \sigma(t)|_{o'\cdot p} \approx \sigma' \circ \sigma(l)|_{o'\cdot p}, \dots, \sigma' \circ \sigma(t_n) \approx \sigma' \circ \sigma(l_n), \sigma' \circ \sigma(E'), \sigma' \circ \sigma(E),$$

und $\sigma' \circ \sigma$ unifiziert alle $l|_q$ und $t|_q$ für $q \in Left(o' \cdot p)$ (beachte $dom(\sigma) \cap dom(\sigma') = \emptyset$ wegen der Linkslinearität der verwendeten Regel).

□

Um den Zusammenhang mit Mustern in einem LR-DT herzustellen, wird eine Eigenschaft von diesen Bäumen benötigt, nämlich daß Muster von Regelknoten für bestimmte Positionen identisch sind mit den Mustern von darüberliegenden *branch*-Knoten (Verzweigung an Position o). Die Identität gilt für alle die Positionen, die links von oder überhalb o liegen. Dazu wird eine Menge *Rule* benötigt, die aus allen Regelknoten besteht, die unterhalb eines *branch*-Knoten liegen.

Definition 3.2.6. ($Rule(v)$)
Die Menge $Rule(v)$ enthält alle Regelknoten, die unterhalb eines branch-Knotens v liegen:

- $Rule(exempt(\pi)) := \emptyset$
- $Rule(rule(\pi \to r)) := \{rule(\pi \to r)\}$
- $Rule(branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_n)) := \bigcup_{1 \le i \le n} Rule(\mathcal{T}_i)$.

△

Im folgenden werden *exempt*-Knoten nicht mehr berücksichtigt.

Lemma 3.2.7.
Es sei $v = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_n)$ ein Knoten in einem LR-DT. Dann gilt für alle $rule(\pi_R \to r) \in Rule(v)$: $Head(\pi_R|_q) = Head(\pi|_q)$ für alle $q \prec o$ und $\pi_R|_q = \pi|_q$ für alle $q \in Left(o)$.

Beweis
durch strukturelle Induktion über den definierenden Baum.
Induktionsverankerung: $v = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_n)$ hat als direkten Nachfolger einen $rule(\pi_R \to r)$-Knoten:

Dann ist $\pi_R = \pi[t]_o$ für einen Term t. Also gilt für alle $q \in \mathcal{O}(\pi_R) \cap \mathcal{O}(\pi)$ mit $q \neq o$: $\pi_R|_q = \pi|_q$, und damit gilt insbesondere $Head(\pi_R|_q) = Head(\pi|_q)$ für alle $q \prec o$ und $\pi_R|_q = \pi|_q$ für alle $q \in Left(o)$.

Induktionsschritt: $v = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ hat als direkten Nachfolger einen $branch(\pi', o', \mathcal{T}'_1, \ldots, \mathcal{T}'_{n'})$-Knoten v':
Dann gilt nach Induktionsvoraussetzung für alle $rule(\pi_R \to r) \in Rule(v') \subseteq Rule(v)$: $Head(\pi_R|_q) = Head(\pi'|_q)$ für alle $q \prec o'$ und $\pi_R|_q = \pi'|_q$ für alle $q \in Left(o')$. Es ist $\pi' = \pi[t]_o$ für einen Term t, also ist $\pi'|_q = \pi|_q$ für $q \in \mathcal{O}(\pi) \cap \mathcal{O}(\pi')$ mit $q \neq o$.
Da nach Definition eines LR-DT o' rechts von oder unter o steht (und damit $o \prec o'$ oder $o \in Left(o')$ gilt), ist dann $\pi|_q = \pi_R|_q$ für $q \in Left(o)$ und $Head(\pi_R|_q) = Head(\pi|_q)$ für $q \prec o$.

□

Es ist jetzt möglich, den Zusammenhang von Σ_3, d.h. dem Verzweigen in einem definierenden Baum, mit einer OINC-Sequenz zu zeigen:

Lemma 3.2.8.
Es sei $\Sigma = \lambda(t, \mathcal{T})\Sigma_3$ Teil einer Aufrufsequenz von λ (Σ_3 ggf. leer), $v = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ der letzte von Σ_3 erreichte Knoten und $t \approx s, E$ eine OINC-geeignete Gleichungsmenge mit $s \notin \mathcal{V}ar$.
Dann gibt es für alle $rule(\pi_R \to r) \in Rule(v)$ eine OINC-Sequenz

$$t \approx s, E \rightsquigarrow_{[on]} \rightsquigarrow^*_{[d,v]} \sigma(t)|_o \approx \sigma(\pi_R)|_o, \sigma(E'), \sigma(E),$$

wobei $\sigma = leftmgu^o(\pi, t)$ ist.

Beweis
t und π sind unifizierbar mittels σ nach Voraussetzung (sonst wäre der Knoten beim Abstieg nicht erreicht worden). Weiterhin ist für alle $rule(\pi_R \to r) \in Rule(v)$ $Head(\pi_R|_q) = Head(\pi|_q)$ für alle $q \prec o$ und $\pi_R|_q = \pi|_q$ für alle $q \in Left(o)$ nach Lemma 3.2.7.
Also sind wegen der Linkslinearität der Regeln insbesondere $t|_q$ und $\pi_R|_q$ unifizierbar für alle $q \in Left(o)$, und es ist $Head(t|_q) = Head(\pi_R|_q)$ für alle $q \prec o$.
Damit gibt es nach Lemma 3.2.3 eine OINC-Sequenz

$$t \approx s, E \rightsquigarrow^* \sigma(t)|_o \approx \sigma(\pi_R)|_o, \sigma(E'), \sigma(E),$$

und σ ist der Unifikator der Positionen links von oder überhalb o.

□

Um einen vollständigen NN-Schritt simulieren zu können, müssen noch Aufruffolgen Σ_4 untersucht werden, d.h. es muß Narrowing für einen Unterterm durchgeführt werden.

Dazu wird zunächst beobachtet, daß eine maximale Folge von λ-Aufrufen der Form Σ_3 (maximal in dem Sinn, daß direkt danach Fall R_3^λ nicht noch einmal auftritt) mit einer korrespondierenden OINC-Sequenz zu mindestens einer Gleichung $s \approx t$ führt mit $s \notin \mathcal{V}ar$ und $t \notin \mathcal{V}ar$:

Lemma 3.2.9.
Es sei Σ_3 eine Folge von Fällen R_3^λ in einer λ-Aufruffolge, der sich direkt ein Fall R_4^λ anschließt, $v = branch(\pi, p, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ der letzte von Σ_3 erreichte Knoten, π_R eine Regel für $Head(t)$ mit $rule(\pi_R \to r) \in Rule(v)$, $\sigma = leftmgu^p(t, \pi)$ und $t \approx s, E$ eine OINC-geeignete Gleichungsmenge mit $s \notin \mathcal{V}ar$.
Dann gibt es eine OINC-Sequenz

$$t \approx s, E \leadsto_{[on]} \leadsto^*_{[d,v]} \sigma(t)|_p \approx \sigma(\pi_R)|_p, \sigma(E'), \sigma(E)$$

mit $\sigma(t)|_p \notin \mathcal{V}ar$ und $\sigma(\pi_R)|_p \notin \mathcal{V}ar$.

Beweis
Es gibt eine Sequenz der angegebenen Form nach Lemma 3.2.8.
Wäre nun einerseits $\sigma(t)|_p \in \mathcal{V}ar$, so würde Fall R_4^λ nicht auftreten: Da das Muster $\tilde{\pi}$ aller direkten Nachfolgerknoten von v nach Definition der DT eine Instanz von π ist mit $\tilde{\pi} = \pi[u]_p$ für einen Term u, und weil t mit π unifizierbar ist (sonst wäre der Knoten v nicht erreicht worden), ist folglich $\sigma(t)|_p$ mit $\tilde{\pi}|_p$ unifizierbar (und damit auch $\sigma(t)$ mit $\tilde{\pi}$, weil die Positionen rechts von p nach Definition der LR-DTs Variablen und die Muster linear sind). Also tritt Fall R_4^λ nicht auf.
Andererseits gibt es keine OINC-Sequenz dieses Aussehens, die $\sigma(\pi_R)|_p \in \mathcal{V}ar$ erfüllt: π_R ist Instanz aller Muster der darüberliegenden Knoten; nach Voraussetzung auch Instanz von π und dem Muster $\tilde{\pi}$ eines direkten Nachfolgerknotens von v. Demzufolge wäre im Fall $\sigma(\pi_R)|_p \in \mathcal{V}ar$ auch $\tilde{\pi}|_p \in \mathcal{V}ar$. Damit wäre dann die Bedingung für Fall R_4^λ nicht erfüllt, weil $\sigma(t)|_p$ und $\tilde{\pi}|_p$ unfizierbar wären (und ebenso mit derselben Argumentation wie im ersten Fall $\sigma(t)$ und $\tilde{\pi}$).

□

Bemerkung 3.2.10.
Wenn Σ_3 die leere Folge darstellt, ergibt sich das gleiche Ergebnis. Wie im Induktionsschritt des Beweises von Lemma 3.2.3 werden die ersten variablen Argumente der Muster des definierenden Baums für t mit [v]-Schritten "übersprungen".

Die Kombination von Folgen der Form Σ_3 und R_4^λ entspricht dem Abstieg in einem definierenden Baum bis zu einem *branch*-Knoten, bei dem die Muster der Unterbäume nicht mit dem Term unifiziert werden können, mit dem λ

aufgerufen wurde. An dieser Stelle muß in der entsprechenden OINC-Ableitung ein [on]-Schritt durchgeführt werden:

Lemma 3.2.11.
Für jede λ-Aufruffolge $\lambda(t, \mathcal{T})\Sigma_3 R_4^\lambda$ gibt es bei gegebener OINC-geeigneter Gleichungsmenge $t \approx s, E$ mit $s \notin \mathcal{V}ar$ eine OINC-Sequenz

$$\begin{array}{lll} t \approx s, E & \leadsto_{[on]}\leadsto^*_{[d,v]} & \sigma(t)|_p \approx \sigma(\pi_R)|_p, \sigma(E'), \sigma(E) \\ & \leadsto_{[on]} & \sigma(t)|_{p\cdot 1} \approx \sigma(\pi'_R)|_{p\cdot 1}, \dots, \sigma(t)|_{p\cdot n} \approx \sigma(\pi'_R)|_{p\cdot n}, \\ & & r' \approx \sigma(\pi_R)|_p, \sigma(E'), \sigma(E), \end{array}$$

wobei $\pi_R \to r$ eine Regel für $Head(t)$, $\pi'_R \to r'$ eine Regel für $Head(t|_p)$, $\sigma = leftmgu^p(t, \pi_R)$ und $n = Arity(Head(t|_p))$ ist.

Beweis
folgt aus Lemma 3.2.9.

□

Um nun die einzelnen Folgen Σ_3 (eventuell leer) und R_4^λ zusammenzusetzen, wird - mit dem Ziel, letztendlich eine Korrespondenz zwischen einem NN- und einem OINC-Schritt zu finden - die Funktion λ um ein weiteres Argument erweitert: der absoluten Position im Term, in der die Berechnung von λ gerade stattfindet.

Definition 3.2.12. (λ')
Die Funktion λ' besitzt als Bildbereich die kleinste Menge mit

$$\lambda'(t, p, \mathcal{T}) \ni \begin{cases} (\varepsilon, l \to r, mgu(t, l)) & \text{falls } \mathcal{T} = rule(l \to r); \quad (R_1^{\lambda'}) \\ (\varepsilon, ?, mgu(t, \pi)) & \text{falls } \mathcal{T} = exempt(\pi); \quad (R_2^{\lambda'}) \\ (p', l \to r, \sigma) & \text{falls } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_k), \\ & \quad \exists i: \ t \text{ und } pattern(\mathcal{T}_i) \text{ unifizierbar und} \\ & \quad (p', l \to r, \sigma) \in \lambda'(t, p, \mathcal{T}_i); \quad (R_3^{\lambda'}) \\ (o \cdot p', l \to r, \sigma \circ \tau) & \text{falls } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_k), \\ & \quad \forall i: t \text{ und } pattern(\mathcal{T}_i) \text{ nicht unifizierbar}, \\ & \quad \tau = mgu(t, \pi), \\ & \quad \mathcal{T}' \text{ def. Baum für } \tau(Head(t|_o)) \text{ und} \\ & \quad (p', l \to r, \sigma) \in \lambda'(\tau(t|_o), p \cdot o, \mathcal{T}'). \quad (R_4^{\lambda'}) \end{cases}$$

△

Der erste Aufruf von λ' wird im folgenden stets mit ε als zweitem Argument stattfinden.

Es kann ein λ'-Aufruf bis zum Erreichen eines Regelknotens durch eine OINC-Sequenz simuliert werden:

Lemma 3.2.13.
Es sei $\Sigma = \lambda'(t, \varepsilon, \mathcal{T})\Sigma_3 R_4^{\lambda'}\Sigma_3 R_4^{\lambda'} \dots R_4^{\lambda'}$ *eine* λ'*-Aufruffolge, wobei* $\lambda'(\tau(t'), p \cdot o, \mathcal{T})$ *der letzte Aufruf dieser Folge sei.* $v = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_n)$ *sei der letzte (vor diesem Aufruf) erreichte Knoten.*
Dann gibt es bei gegebener OINC-geeigneter Gleichungsmenge $t \approx s, E$ *mit* $s \notin \mathcal{V}ar$ *eine OINC-Sequenz*

$$t \approx s, E \rightsquigarrow^* \sigma(t)|_{p \cdot o} \approx \sigma(\pi_R)|_o, \sigma(E'), \sigma(E),$$

wobei folgende Bedingungen gelten:

1. $t' = t|_{p \cdot o}$,
2. $rule(\pi_R \to r) \in Rule(v)$,
3. $\sigma = \tau[\mathcal{V}ar(t)]$
4. $\sigma(t)|_{p \cdot o} \notin \mathcal{V}ar, \sigma(\pi_R)|_o \notin \mathcal{V}ar$.

Beweis
durch Induktion über die Länge von Σ.
Induktionsverankerung: $\Sigma = \lambda'(t, \varepsilon, \mathcal{T})\Sigma_3 R_4^{\lambda'}$.
Für den letzten erreichten Knoten $v = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_n)$ und den letzten λ'-Aufruf $\lambda'(\tau(t'), p \cdot o, \mathcal{T})$ ist $p = \varepsilon$ nach Definition von λ' (erst das Auftreten des Falls $R_4^{\lambda'}$ ändert das zweite (Positions-)argument von λ' zu o). Damit folgt $t|_o = t'$. Lemma 3.2.8 liefert eine OINC-Sequenz der angegebenen Form; damit ist auch Bedingung (2) erfüllt, und es ist $\sigma = leftmgu^o(\pi, t)$. Lemma 3.2.9 liefert die Gültigkeit von Bedingung (4). Nach Definition der LR-DTs ist $\pi|_q \in \mathcal{V}ar$ für $q = o$ und alle $o \in Left(q)$. Also ist $mgu(\pi, t) = leftmgu^o(\pi, t)[\mathcal{V}ar(t)]$, und wegen $\tau = mgu(t, \pi)$ ergibt sich $\tau = \sigma[\mathcal{V}ar(t)]$.

Induktionsschritt: $\Sigma = \Sigma'\Sigma_3 R_4^{\lambda'}$.
Σ' führe zu einem Knoten $branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_n)$. Dann ist $\lambda'(\sigma'(t|_{p \cdot o}), p \cdot o, \mathcal{T}')$ der letzte Aufruf von Σ' (Fall $R_4^{\lambda'}$), und es ist $\tau'(t)|_{p \cdot o} \approx \tau'(\pi_R)|_o, \tau'(E'), \tau'(E)$ die von OINC inferierte Gleichungsmenge nach Induktionsvoraussetzung mit $\tau' = \sigma'[\mathcal{V}ar(t)]$. Schließlich sei $\tilde{v} = branch(\tilde{\pi}, \tilde{o}, \widetilde{\mathcal{T}}_1, \dots, \widetilde{\mathcal{T}}_{\tilde{n}})$ der letzte von Σ erreichte Knoten und $\lambda'(t', p', \mathcal{T}'_1, \dots, \mathcal{T}'_{n'})$ der letzte λ'-Aufruf von Σ. Die auf Σ' folgenden $R_3^{\lambda'}$-Fälle führen mit dem ersten Aufruf $\lambda'(\sigma'(t|_{p \cdot o}), p \cdot o, \mathcal{T}')$ nach Lemma 3.2.8 zu

$$\tau \circ \tau'(t)|_{p \cdot o \cdot \tilde{o}} \approx \tau \circ \tau'(\tilde{\pi}_R)|_{\tilde{o}}, \tau \circ \tau'(E'), \tau \circ \tau'(E).$$

mit $\tau = leftmgu^{\tilde{o}}(\tilde{\pi}, \tau'(t)|_{p \cdot o})$ und $rule(\tilde{\pi}_R \to r) \in Rule(\tilde{v})$ (womit die Gültigkeit von Bedingung (2) nachgewiesen ist).

Zu zeigen bleibt, daß die restlichen drei Bedingungen erfüllt sind.
Es ist $p' = p \cdot o \cdot \tilde{o}$ nach Definition von λ'; Auftreten des Falls $R_3^{\lambda'}$ ändern dieses Argument nicht. Also ist $t' = \sigma(t)|_{p'}$ mit $\sigma = mgu(\tilde{\pi}, \sigma'(t)|_{p'}) \circ \sigma'$ nach Definition von λ'; Bedingung (1) gilt.
Weiterhin ist $\tau' = \sigma'[\mathcal{V}ar(t)]$ nach Induktionsvoraussetzung;
zu zeigen ist $leftmgu^{\tilde{o}}(\tilde{\pi}, \tau'(t)|_{p \cdot o}) \circ \tau' = \big(mgu(\tilde{\pi}, \sigma'(t)|_{p \cdot o \cdot \tilde{o}}) \circ \sigma'\big)[\mathcal{V}ar(t)]$.
Da die Positionen in $\tilde{\pi}$ rechts von $\tilde{o}$ mit frischen Variablen besetzt sind, folgt aus der Linearität der Muster durch Einsetzen $\tau \circ \tau' = \sigma \circ \sigma'[\mathcal{V}ar(t)]$; Bedingung (3) gilt.
Bedingung (4) schließlich ist gültig, weil der letzte Aufruf von Σ nach Voraussetzung ein Fall $R_4^{\lambda'}$ ist (vgl. die Beweisführung in Lemma 3.2.9).

□

Lemma 3.2.13 ermöglicht nun den Beweis des zentralen Satzes dieses Abschnitts.

Satz 3.2.14.

1. *Wenn ein NN-Schritt $t \rightsquigarrow_{p,l\to r,\sigma} t'$ durchgeführt werden kann, dann gibt es für eine OINC-geeignete Gleichungsmenge $t \approx s, E$ mit $s \notin \mathcal{V}ar$ eine OINC-Sequenz*

$$\begin{aligned} t \approx s, E \rightsquigarrow^* \tau(t)|_p \approx \tau(u), \tau(E'), \tau(E) \\ \rightsquigarrow_{[on]} \rightsquigarrow^*_{[d,v]} \tau'(r) \approx \tau' \circ \tau(u), \tau' \circ \tau(E'), \tau' \circ \tau(E) \end{aligned}$$

   mit $\tau' = mgu(\tau(t)|_p, l)$, $\tau' \circ \tau = \sigma[\mathcal{V}ar(t)]$, $u = s$, falls $p = \varepsilon$, und $u = \pi_R|_o$ sonst, wobei π_R die linke Seite der im letzten [on]-Schritt angewandten Regel ist und o die Verzweigungsposition desjenigen Knotens, in dem Fall $R_4^{\lambda'}$ zuletzt auftrat.

2. *Wenn bei Errechnung der Substitution σ n-mal der Fall $R_4^{\lambda'}$ auftritt, dann wird in der entsprechenden OINC-Sequenz $n+1$-mal eine [on]-Inferenz durchgeführt.*

Beweis von (1) durch Fallunterscheidung über die λ'-Aufruffolge der Form $\Sigma = \lambda'(t, \varepsilon, \mathcal{T}) \cdots R_1^{\lambda'}$ mit $t = f(t_1, \ldots, t_n)$.
1.Fall: $\Sigma = \lambda'(t, \varepsilon, \mathcal{T}) R_1^{\lambda'}$.
Dann besteht $\mathcal{T}$ nur aus einem $rule(f(l_1, \ldots, l_n) \to r)$-Knoten. Also gibt es einen OINC-Schritt

$$t \approx s, E \rightsquigarrow_{[on]} t_1 \approx l_1, \ldots, t_n \approx l_n, r \approx s, E.$$

Da t und $f(l_1, \ldots, l_n)$ unifizierbar sind mit mgu σ (sonst könnte die Regel nicht angewendet werden), gibt es nach Lemma 3.2.1 eine OINC-Sequenz

$$t \approx s, E \rightsquigarrow^* \sigma(r) \approx \sigma(s), \sigma(E),$$

und es ist $\sigma \circ \emptyset = \sigma$.

2.Fall: $\Sigma = \lambda'(t, \varepsilon, \mathcal{T})\Sigma_3 R_1^{\lambda'}$.
Da $R_4^{\lambda'}$ nicht in Σ auftritt, entspricht dieser Fall dem vollständigen Abstieg in *einem* DT. Wenn π das Muster letzten erreichten *branch*-Knoten v bezeichnet, dann ist π mit t unifizierbar (und damit auch alle l für $rule(l \to r) \in Rule(v)$). Der Beweis erfolgt dann analog zu Fall 1.

3.Fall:
Dann gibt es in Σ ein oder mehrere Auftreten des Falls $R_4^{\lambda'}$. Der bei Auftreten dieses Falls erreichte Knoten sei $v = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$. Dann gibt es nach Lemma 3.2.13 eine OINC-Sequenz

$$t \approx s, E \rightsquigarrow^* \tau(t)|_{p \cdot o} \approx \tau(\pi'_R)|_o, \tau(E'), \tau(E)$$

mit $\tau = \sigma'[\mathcal{V}ar(t)]$ und $rule(\pi'_R \to r') \in Rule(v)$, wenn $\lambda'(\sigma'(t|_{p \cdot o}), p \cdot o, \mathcal{T})$ der Aufruf bei Auftreten des Falls $R_4^{\lambda'}$ ist.
Lemma 3.2.11 liefert die Möglichkeit der Anwendung eines [on]-Schritts mit einer Regel $l \to r$, deren linke Seite mit $\tau(t)|_{p \cdot o}$ unifizierbar ist mittels eines Unifikators τ' (sonst könnte Fall $R_1^{\lambda'}$ am Ende der Sequenz nicht auftreten):

$$\ldots \rightsquigarrow_{[on]} \quad \tau(t|_{p \cdot o \cdot 1}) \approx l|_1, \ldots, \tau(t|_{p \cdot o \cdot n}) \approx l|_n, r \approx \tau(\pi'_R)|_o, \tau(E'), \tau(E).$$

mit $n = Arity(Head(l))$.
Aus der Unifizierbarkeit von $\sigma'(t)|_{p \cdot o}$ mit l folgt wieder analog zum 1. Fall das gewünschte Ergebnis (mit τ anstelle von $\emptyset$) mit der resultierenden Substitution $\sigma = (\tau' \circ \tau)[\mathcal{V}ar(t)]$.

Beweis von 2:
Eine [on]-Inferenz wird beim initialen Aufruf von λ' durchgeführt. Aus Lemma 3.2.8 folgt, daß in einer Teilsequenz Σ_3 keine [on]-Inferenzen durchgeführt werden müssen; Lemma 3.2.9 liefert eine [on]-Inferenz für jedes Auftreten von $R_4^{\lambda'}$.

□

Die Korrespondenz erfolgreicher Ableitungen ("einer NN-Ableitung entspricht genau eine OINC-Ableitung") wird in den folgenden Abschnitten untersucht. Dazu wird zunächst aus einer NN-Ableitung eine OINC-Ableitung konstruiert und danach gezeigt, daß OINC und NN bestimmte Eindeutigkeitseigenschaften besitzen, woraus die gegenseitige Simulationsbeziehung folgt.

3.3 Vergleich erfolgreicher Ableitungen

Im nächsten Abschnitt wird zuerst eine NN-Ableitung durch eine OINC-Ableitung simuliert. Es folgt der Nachweis, daß OINC (wie auch NN) Lösungen nicht mehrfach berechnet und daß die Lösungen unabhängig voneinander sind, womit die wechselseitige Simulationsbeziehung nachgewiesen wird.

3.3.1 Simulation von NN durch OINC

Wenn eine [on]-Inferenz in einem Term $f(t_1, \dots, t_n)$ durchgeführt wird, dann wird zunächst versucht, die Argumente $t_1, \dots, t_n$ mit den Argumenten $l_1, \dots, l_n$ einer linken Regelseite $f(l_1, \dots, l_n)$ modulo dem betrachteten TES $\mathcal{R}$ zu unifizieren. Wenn diese Unifikation modulo $\mathcal{R}$ erfolgreich war, wird die errechnete Substitution auf die rechte Regelseite angewendet, und das Verfahren wird für diese rechte Regelseite iteriert.

NN verhält sich im Prinzip ganz genauso: Erst wenn alle Argumente $t_1, \dots, t_n$ des betrachteten Terms mit den Argumenten der linken Regelseite modulo $\mathcal{R}$ unifiziert sind, wird das Narrowing an der Wurzelposition des betrachteten Terms durchgeführt.

Um diese Beobachtung modellieren und so einen Zusammenhang zwischen OINC- und NN-Ableitungen herstellen zu können, werden Informationen über den zu ersetzenden Teilterm, die Position der Ersetzung und die Anzahl der Funktionssymbole über der Ersetzungsposition protokolliert. Ein Beispiel folgt nach den beiden folgenden Definitionen.

Definition 3.3.1. *(OINC')*
OINC' ist durch folgendes Inferenzschema definiert:

- *[in], Initialisierung*

$$\frac{t_1 \approx s_1, E}{[t_1, T_{1,1}, \varepsilon, 0] \approx [s_1, T_{1,2}, \varepsilon, 0], E}$$

  mit $\{t_1, s_1\} \subseteq \mathcal{T}(\Sigma, \mathcal{X})$*;*

- *[on],*

$$\frac{[f(t_1, \dots, t_k), U, p, n] \approx [s, S, q, m], E}{A_1 \approx B_1, \dots, A_k \approx B_k, [r, R_j, p, n+1] \approx [s, S, q, m], E}$$

  mit $A_i = [t_i, U, p \cdot i, n+1]$ *und* $B_i = [l_i, L_j, p \cdot i, n+1]$ *für* $1 \leq i \leq k$ *bei Anwendung der (frischen)* j*-ten Regel* $f(l_1, \dots, l_k) \to r$, $U \in \{T_{m,1}, L_m, R_m\}$ *und* $s \notin \mathcal{V}ar$.

- *[d],*

$$\frac{[f(t_1, \dots, t_k), T, p, n] \approx [f(s_1, \dots, s_k), S, q, m], E}{[t_1, T, p \cdot 1, n] \approx [s_1, S, q \cdot 1, m], \dots, [t_k, T, p \cdot k, n] \approx [s_k, S, q \cdot k, m], E};$$

- *[v],*

$$\frac{[t,T,p,n] \approx [x,S,q,m], E}{\vartheta(E)} \text{ und } \frac{[x,S,q,n] \approx [t,T,p,m], E}{\vartheta(E)},\ t \notin \mathcal{V}ar$$

mit $\vartheta = \{x \mapsto t\}$ *und* $x \in \mathcal{V}ar$.

Dabei gilt:

1. *[in] ist der initiale Schritt, der aus einer Gleichung die Gleichung aus Quadrupeln erzeugt.*

2. *Die erste Komponente ist der Term, in dem das Narrowing durchgeführt wird. Sie entspricht den linken bzw. rechten Gleichungsseiten, die von OINC inferiert werden. Damit ist dieser Term Teil einer initialen Gleichungsseite oder einer Regelseite.*

3. *Die zweite Komponente ist ein Symbol, das für die Art des Terms steht, in dem Narrowing durchgeführt wird. Das kann eine Regelseite oder eine Gleichungsseite einer initialen Gleichung sein. Dieses Symbol ist Element der Menge*

$$\bigcup_{i=1}^{n}\{L_i, R_i\} \cup \bigcup_{j=1}^{m}\{T_{j,1}, T_{j,2}\}$$

bei insgesamt n *Regeln und* m *initialen Gleichungen. Dabei steht* L_i *für eine linke Regelseite,* R_i *für eine rechte Regelseite und* $T_{j,1}$ *für die linke und* $T_{j,2}$ *für die rechte Seite der* j*-ten initialen Gleichung. Die* R_i *kommen in inferierten Gleichungen* $A \approx B$ *nur auf linken Gleichungsseiten* A *vor.*

4. *Die dritte Komponente repräsentiert die Position, in der das Narrowing durchgeführt wird.*

5. *Die vierte Komponente repräsentiert die Anzahl der Funktionssymbole, die über der Narrowingposition stehen. Die eigentliche Intention ist, jeden Narrowingschritt mit einer eindeutigen Nummer zu indizieren, wozu man aber eine "globale" Variable benötigte. Für die hier verfolgten Ziele ist die Anzahl der Funktionssymbole über der Narrowingposition ausreichend.*

6. *Substitutionen werden kanonisch auf die Quadrupel erweitert, indem sie nur die erste Komponente modifizieren.*

$\triangle$

Sämtliche Resultate für OINC lassen sich selbstverständlich auf OINC' übertragen.

Definition 3.3.2. *([on]-gelöst, [on]-n-gelöst)*
Es sei G eine OINC-geeignete Gleichungsmenge, σ eine Substitution und

$$\begin{array}{rl}
G \rightsquigarrow^{*} & [t,U,p,n-1] \approx [t',U',p',n'],G' \\
\rightsquigarrow_{[on]} & F,[r,R,p,n] \approx [t',U',p',n'],G' \\
\rightsquigarrow^{*} & E \\
\rightsquigarrow^{+} & \sigma\big([r,R,p,n] \approx [t',U',p',n'],G'\big) \\
\rightsquigarrow^{+} & \square
\end{array}$$

eine OINC'-Ableitung, so daß für alle möglichen E gilt: Es gibt keine Substitution ϑ mit $E = \vartheta\big([r,R,p,n] \approx [t',U',p',n'],G'\big)$, d.h. die angegebene Gleichungsmenge $E' := \sigma\big([r,R,p,n] \approx [t',U',p',n'],G'\big)$ ist die erste inferierte dieses Aussehens.
Dann heißt E' [on]-n-gelöst bzgl. der angegebenen Inferenz (oder auch nur [on]*-gelöst). Die angegebene [on]-Inferenz heißt in dem Moment gelöst, in dem E' inferiert wird.*

$\triangle$

Eine [on]-Inferenz heißt also gelöst, falls eine Instanz der rechten Seite der angewendeten Regel das erste Mal als linke Seite der linkesten Gleichung einer inferierten Gleichungsmenge E auftaucht. E heißt dann [on]-gelöst bzgl. der betrachteten [on]-Inferenz.

Beispiel 3.3.3. *(OINC', [on]-gelöst)*
Es sei $\mathcal{R} = \{h(d) \to c, g(c) \to b, f(b) \to a\}$ ein TES (dabei seien die Regeln geordnet, d.h. die erste Regel hat den Index 1, die zweite den Index 2 und die dritte den Index 3).
Dann gibt es für die initiale Gleichung $f(g(h(x))) \approx a$ eine OINC'-Sequenz

$$\begin{array}{rll}
f(g(h(x))) \approx a & \rightsquigarrow_{[in]} & [f(g(h(x))),T_{1,1},\varepsilon,0] \approx [a,T_{1,2},\varepsilon,0] \\
& \rightsquigarrow_{[on1]} & [g(h(x)),T_{1,1},1,1] \approx [b,L_3,1,1], \\
& & [a,R_3,\varepsilon,1] \approx [a,T_{1,2},\varepsilon,0] \\
& \rightsquigarrow_{[on2]} & [h(x),T_{1,1},1\cdot 1,2] \approx [c,L_2,1\cdot 1,2], \\
& & [b,R_2,1,2] \approx [b,L_3,1,1], \\
& & [a,R_3,\varepsilon,1] \approx [a,T_{1,2},\varepsilon,0] \\
& \rightsquigarrow_{[on3]} & [x,T_{1,1},1\cdot 1\cdot 1,3] \approx [d,L_1,1\cdot 1\cdot 1,3], \\
& & [c,R_1,1\cdot 1,3] \approx [c,L_2,1\cdot 1,2], \\
& & [b,R_2,1,2] \approx [b,L_3,1,1], \\
& & [a,R_3,\varepsilon,1] \approx [a,T_{1,2},\varepsilon,0] \\
\Longrightarrow^{3} & \rightsquigarrow_{[v]} & [c,R_1,1\cdot 1,3] \approx [c,L_2,1\cdot 1,2], \\
& & [b,R_2,1,2] \approx [b,L_3,1,1], \\
& & [a,R_3,\varepsilon,1] \approx [a,T_{1,2},\varepsilon,0] \\
\Longrightarrow^{2} & \rightsquigarrow_{[d]} & [b,R_2,1,2] \approx [b,L_3,1,1], \\
& & [a,R_3,\varepsilon,1] \approx [a,T_{1,2},\varepsilon,0] \\
\Longrightarrow^{1} & \rightsquigarrow_{[d]} & [a,R_3,\varepsilon,1] \approx [a,T_{1,2},\varepsilon,0] \\
& \rightsquigarrow_{[d]} & \square
\end{array}$$

Die bei $\Longrightarrow^3$ inferierte Gleichungsmenge ist [on]-gelöst bzgl. des dritten [on]-Schritts, die bei $\Longrightarrow^2$ inferierte Gleichungsmenge ist [on]-gelöst bzgl. des zweiten [on]-Schritts, und die bei $\Longrightarrow^1$ inferierte Gleichungsmenge ist [on]-gelöst bzgl. des ersten [on]-Schritts. Die drei Pfeile geben auch die Zeitpunkte der [on]-Lösungen an.
Sieht man sich zum Vergleich die entsprechende NN-Sequenz

$$f(g(\underline{h}(x))) \rightsquigarrow f(\underline{g}(c)) \rightsquigarrow \underline{f}(b) \rightsquigarrow a$$

an, wobei die Narrowingpositionen unterstrichen sind, so läßt sich feststellen, daß die [on]-Inferenzen zwar an denselben Positionen, aber in unterschiedlicher Reihenfolge angewendet werden. Die Motivation für die Einführung des Begriffs "[on]-gelöst" liegt nun darin, daß die [on]-Inferenzen in derselben Reihenfolge gelöst werden, wie die entsprechenden NN-Schritte durchgeführt werden.

■

Zunächst wird nun das Ergebnis von Satz 3.2.14 auf OINC' übertragen. Dazu sei im folgenden wiederum vorausgesetzt, daß die Modellierung der Regeln mit LR-DTs erfolgt.

Satz 3.3.4.
Wenn ein NN-Schritt $t \rightsquigarrow_{p,l\to r,\sigma} t'$ durchgeführt werden kann und i der Index der angewandten Regel ist, dann gibt es bei gegebener OINC-geeigneter Gleichungsmenge $t \approx s, E$ mit $s \notin \mathcal{V}ar$ eine OINC'-Sequenz

$$t \approx s, E \rightsquigarrow^* [\tau'(r), R_i, p, n] \approx [\tau' \circ \tau(u), X, q, m], \tau' \circ \tau(E'), \tau' \circ \tau(E)$$

mit $\tau' = mgu(l, \tau(t|_p))$ und $\tau' \circ \tau = \sigma[\mathcal{V}ar(t)]$, wobei $[\tau'(r), R_i, p, n] \approx [\tau' \circ \tau(u), X, q, m], \tau' \circ \tau(E'), \tau' \circ \tau(E)$ [on]-n-gelöst ist.

Beweis
Wie im Beweis von Satz 3.2.14 sind drei Fälle der Form der λ'-Aufruffolge Σ zu unterscheiden. Hier wird exemplarisch nur der erste Fall untersucht (die anderen ergeben sich sofort, wenn man die OINC- durch OINC'-Ableitungen ersetzt.
t sei wie dort von der Form $f(t_1, \ldots, t_n)$ und $l \to r$ eine Regel mit Index i für f mit $l = f(l_1, \ldots, l_n)$.
1.Fall: $\Sigma = \lambda'(t, \varepsilon, \mathcal{T})R_4^{\lambda'}$
Die entsprechende OINC'-Sequenz ist von der Form

$$\begin{array}{rll} t \approx s, E & \rightsquigarrow_{[in]} & [t, T_{1,1}, \varepsilon, 0] \approx [s, T_{1,2}, \varepsilon, 0], E \\ & \rightsquigarrow_{[on]} & [t_1, T_{1,1}, 1, 1] \approx [l_1, L_i, 1, 1], \ldots, [t_n, T_{n,1}, n, 1] \approx [l_n, L_i, n, 1], \\ & & [r, R_i, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0], E \\ & \rightsquigarrow^*_{[d,v]} & [\sigma'(r), R_i, \varepsilon, 1] \approx [\sigma'(s), T_{1,2}, \varepsilon, 0], \sigma'(E'), \sigma'(E) \end{array}$$

mit $\sigma' = mgu(t, l)$ nach Lemma 3.2.1. Da die angewandten Regeln in der NN- und der OINC'-Sequenz gleich bis auf Umbenennung der Variablen in den Regeln bzw. den Mustern der DT-Knoten sind, folgt $\sigma = \sigma'[\mathcal{V}ar(t)]$. Die letzte Gleichung ist trivialerweise [on]-1-gelöst.
Die beiden anderen Fälle werden exakt wie im Beweis des Satzes 3.2.14 behandelt. Wie im ersten Fall ist es offensichtlich, daß die ersten zwei Komponenten die "richtige" rechte Regelseite repräsentieren. Die resultierende Gleichungsmenge ist im zweiten Fall ebenso trivialerweise [on]-1-gelöst wie im ersten Fall. Im dritten Fall ist die resultierende Gleichungsmenge [on]-n-gelöst, weil der [on]-Schritt ausgehend von einer Gleichungsmenge E ausgeführt wurde, die keine Vorkommen von n als vierte Komponente in beiden Gleichungsseiten aller Gleichungen enthält. Der letzte (n-te) [on]-Schritt erzeugt eine solche Komponente. Die Unifizierbarkeit der linken Regelseite mit dem Term an der Narrowingposition liefert dann (s. Beweis Satz 3.2.14) die Behauptung, daß die resultierende Gleichungsmenge [on]-n-gelöst ist.

□

Im Beweis von Satz 3.2.14 wird eine OINC-Sequenz ausgehend von einem Term t konstruiert. Für den ersten NN-Schritt $t \rightsquigarrow t'$ ist das auch ausreichend. Bei den folgenden NN-Schritten ergibt sich die Schwierigkeit, daß das Narrowing nicht mehr im initialen Term t durchgeführt wird, sondern in t'. Um die entsprechenden OINC-Sequenzen konkatenieren zu können, wird eine wesentliche Eigenschaft der Funktion λ' benötigt: Die λ'-Aufruffolgen zweier direkt aufeinanderfolgender Schritte besitzen nämlich ein gemeinsames Präfix, wobei die errechnete Substitution des zweiten Präfixes (höchstens) eine Variablenumbenennung ist (vgl. die Implementierung in [Han95] und die Konstruktion mit *case*-Ausdrücken in [HP96]). Dieses Präfix ist identisch bis zu demjenigen Funktionssymbol, das über den beiden Narrowingpositionen liegt (und gleich einer dieser beiden Positionen sein kann).

Definition 3.3.5. *($Pos_\top$)*
$Pos_\top(q_1, q_2, t_1, t_2)$ bezeichnet für Positionen p, q_1, q_2 und für Terme t_1, t_2, r mit $t_2 = \sigma(t_1[r]_p)$ die tiefste gemeinsame Position eines Funktionssymbols in t_1 und t_2, die über q_1, q_2 und p liegt, d.h. es gilt $q = Pos_\top(q_1, q_2, t_1, t_2)$ genau dann, wenn $t_2 = \sigma(t_1[r]_p), q \in \mathcal{O}(t_1) \cap \mathcal{O}(t_2), q \preceq p$ und $Head(t_1|_q) \in \mathcal{F}$ ist, wobei es kein q' mit $q \prec q' \prec p, Head(t_1|_{q'}) \in \mathcal{F}$ und $Head(t_2|_{q'}) \in \mathcal{F}$ gibt.

△

Beispiel 3.3.6. *($Pos_\top$)*
Für $t_1 = f(a, f(b, g(c)))$ und $t_2 = f(a, f(b, d))$ ist

- *$Pos_\top(2, 2, t_1, t_2) = 2$,*
- *$Pos_\top(2 \cdot 2, 2 \cdot 2, t_1, t_2) = 2$ und*
- *$Pos_\top(1, 2 \cdot 2, t_1, t_2) = \varepsilon$.*

■

Lemma 3.3.7.
In einer NN-Sequenz der Form

$$t_1 \rightsquigarrow_{p,l\to r,\sigma} t_2 \rightsquigarrow_{p',l'\to r',\sigma'} t_3 \rightsquigarrow^* k$$

für einen Term k in Kopfnormalform gilt für die Präfixe Δ_1 und Δ_2 der zugehörigen λ'-Aufruffolgen
$\Sigma_1 =$
$\underbrace{\lambda'(t_1^{(1)}, p^{(1)}, \mathcal{T}^{(1)}) \cdots \lambda'(t_1^{(n)}, p^{(n)}, \mathcal{T}^{(n)})}_{\Delta_1} \lambda'(t_1^{(n+1)}, q, \mathcal{T}^{(n+1)}) \cdots \lambda'(t_1^{(m)}, p^{(m)}, \mathcal{T}^{(m)})$
und
$\Sigma_2 =$
$\underbrace{\lambda'(t_2^{(1)}, \tilde{p}^{(1)}, \tilde{\mathcal{T}}^{(1)}) \cdots \lambda'(t_2^{(\tilde{n})}, \tilde{p}^{(\tilde{n})}, \tilde{\mathcal{T}}^{(\tilde{n})})}_{\Delta_2} \lambda'(t_2^{(\tilde{n}+1)}, q, \tilde{\mathcal{T}}^{(\tilde{n}+1)}) \cdots \lambda'(t_2^{(\tilde{m})}, \tilde{p}^{(\tilde{m})}, \tilde{\mathcal{T}}^{(\tilde{m})})$
mit $t_1^{(1)} = t_1,\ t_2^{(1)} = t_2,\ p^{(1)} = \tilde{p}^{(1)} = \varepsilon$ und $q = Pos_\top(t_1, t_2, p, p')$:

1. *$\tilde{n} = n$.*
2. *Für jedes Element $\lambda'(t_2^{(i)}, \tilde{p}^{(i)}, \tilde{\mathcal{T}}^{(i)})$ von Δ_2 mit $1 < i \leq n$ ist $t_2^{(i)} = t_2|_{\tilde{p}^{(i)}}$ (bis auf Variablenumbenennung), $p^{(i)} = \tilde{p}^{(i)}$ und $\mathcal{T}^{(i)} = \tilde{\mathcal{T}}^{(i)}$.*
3. *Die besuchten Knoten der entsprechenden LR-DTs sind identisch bis auf Variablenumbenennung und werden in identischer Reihenfolge besucht.*

Dabei seien Δ_1 und Δ_2 insofern minimal, als die λ'-Aufrufe mit q als zweitem Argument den jeweils ersten solchen Aufruf darstellen.

Beweis
durch Induktion über die Länge des Präfixes Δ_1. Dabei werden die Existenz von Δ_2 und die Übereinstimmung simultan gezeigt.
Induktionsverankerung: $\Delta_1 = \emptyset$.
Dann ist $q = \varepsilon$, und die Aussage ist trivial.

Induktionsschritt von $n-1$ nach n: $\Delta_1 = \Gamma_1 \lambda'(t_1^{(n)}, p^{(n)}, \mathcal{T}^{(n)})$.
Es sei $\Delta_2 = \Gamma_2 \lambda'(t_2^{(n)}, \tilde{p}^{(n)}, \tilde{\mathcal{T}}^{(n)})$.
Nach Induktionsvoraussetzung führen die letzten λ'-Aufrufe von Γ_1 und Γ_2 zu einem (modulo Variablenumbenennung identischen) Knoten v mit Muster π und Verzweigungsposition o.
1. Fall:
Angenommen, der letzte Aufruf von Δ_1, $\lambda'(\eta^{(n)}(t_1)|_{p^{(n)}}, p^{(n)}, \mathcal{T}^{(n)})$, ist das Auftreten eines Falls $R_3^{\lambda'}$ (der Aufruf hat diese Form nach Definition von λ').
Dann sind $\eta^{(n)}(t_1)|_{p^{(n)}}$ und π' unifizierbar, wobei π' das Muster eines direkten Nachfolgerknotens von v ist. Damit folgt aus der Definition definierender Bäume:

$$\pi' = \pi[c(x_1, \ldots, x_k)]_o \text{ mit } c \in \mathcal{C}, k \geq 0, x_i \in \mathcal{V}ar \text{ für } 1 \leq i \leq k.$$

Es ist nun zunächst zu zeigen, daß $t_2|_{p^{(n)}}$ und $\varrho(\pi')$ für eine Variablenumbenennung ϱ (des kompletten definierenden Baums; frische Variablen!) unifizierbar sind (damit wird die *Möglichkeit* eines identischen Schritts in Γ_2 nachgewiesen).
Aus der Unfizierbarkeit von π' und $\eta^{(n)}(t_1)|_{p^{(n)}}$ folgt insbesondere die Unifizierbarkeit von $\eta^{(n)}(t_1)|_{p^{(n)}\cdot o}$ mit $c(x_1, \dots, x_k)$. Also ist entweder

$$\eta^{(n)}(t_1)|_{p^{(n)}\cdot o} \in \mathcal{V}ar \text{ oder } Head(\eta^{(n)}(t_1)|_{p^{(n)}\cdot o}) = c.$$

Im ersten Fall wird $\{\eta^{(n)}(t_1|_{p^{(n)}\cdot o}) \mapsto c(x_1, \dots, x_k)\}$ von σ nach Definition von λ' wegen der Verwendung linkslinearer Regeln subsumiert; daraus folgt die Unifizierbarkeit von $t_2|_{p^{(n)}\cdot o}$ mit $\varrho(c(x_1, \dots, x_k))$ (denn durch die Anwendung von σ auf t_1 ist $Head(t_2|_{p^{(n)}\cdot o}) = c$). Im zweiten Fall muß auch $Head(t_2|_{p^{(n)}\cdot o}) = c$ sein (σ verändert nur variable Teilterme, und die Narrowingposition p liegt unter $p^{(n)}$). Da $\varrho(\pi)$ und $t_2|_{p^{(n)}}$ unifizierbar sind, $\varrho(\pi')$ und $\varrho(\pi)$ an Positionen links von und über o identisch sind (Lemma 3.2.7), in $\varrho(\pi')$ rechts von o nur Variablen vorkommen und $p^{(n)} \prec q$ nach Definition von λ' ist, sind wegen der Linearität der Muster dann $t_2|_{p^{(n)}}$ und $\varrho(\pi')$ unifizierbar.
Im Fall $R_3^{\lambda'}$ kann aber Nichtdeterminismus auftreten. Es sei also $t_2|_{p^{(n)}}$ zusätzlich mit π'' unifizierbar, wobei $\pi'' \neq \pi'$ das Muster eines weiteren Nachfolgerknotens von v sei. Dies ist nicht möglich, weil $\sigma \geq mgu(\pi', t_1|_{p^{(n)}})$ ist und nach Lemma 2 in [AEH94b] die Muster einer Ebene des Baums nicht miteinander unifizierbar sind.
Zu zeigen bleibt $t_2^{(n)}|_{p^{(n)}} = t_2^{(n-1)}|_{p^{(n-1)}}$ (dabei sei $n-1$ der Index des letzten Aufrufs von Γ_1). Das folgt aus der Tatsache, daß bei einem Auftreten von Fall $R_3^{\lambda'}$ nur das dritte Argument λ' (nämlich der definierende Baum, in den abgestiegen wird) modifiziert wird. Dieses dritte Argument ist in beiden Fällen natürlich identisch, weil mit demselben Muster unifiziert wurde.
Da es keinen Nichtdeterminismus zwischen den einzelnen Regeln der Definition von λ' gibt, ist dieser Schritt der einzig mögliche. Es wird also der bis auf ϱ identische Knoten erreicht.

2. Fall:
$\lambda'(\eta^{(n)}(t_1)|_{p^{(n)}}, p^{(n)}, \mathcal{T}^{(n)})$ ist das Auftreten eines Falls $R_4^{\lambda'}$.
Dann ist $Head(\eta^{(n)}(t_1)|_{p^{(n)}\cdot o}) \in \mathcal{F}$. Angenommen, es ist $Head(\eta^{(n)}(t_1)|_{p^{(n)}\cdot o}) \neq Head(t_2|_{p^{(n)}\cdot o})$. Dann müßte wegen $p^{(n)} \prec q \preceq \{p, p'\}$ auch $Head(\sigma(t_1)|_{p^{(n)}\cdot o}) \neq Head(t_1|_{p^{(n)}\cdot o})$ sein, was nicht sein kann, da die Positionen über $p^{(n)} \cdot o$ nicht variabel sind (sonst wäre $p^{(n)}$ nicht erreicht worden). Für die letzten erreichten Knoten mit Muster π bzw. $\varrho(\pi)$ für eine Variablenumbenennung ϱ wird durch den angegebenen λ'-Aufruf wiederum derselbe Knoten erreicht; das dritte Argument beider λ'-Aufrufe ist identisch. $p^{(n)} = p^{(n-1)}$ ergibt sich aus der Induktionsvoraussetzung (die beiden Knoten mit Muster π bzw. $\varrho(\pi)$ besitzen dieselbe Verzweigungsposition). σ subsumiert den Unifikator mit dem Muster π; daraus folgt $t_2^{(n)}|_{p^{(n)}} = t_2^{(n-1)}|_{p^{(n-1)}}$.

Es bleibt jetzt noch zu zeigen, daß der abschließende Schritt an Position q identisch ist. Bei diesem Schritt muß es sich wegen $q \neq p^{(n)}$ um das Auftreten eines Falls $R_4^{\lambda'}$ handeln; damit folgt die Übereinstimmung beider Schritte analog zur Beweisführung im 2. Fall.
$n = \tilde{n}$ folgt aus der eineindeutigen Zuordnung der Schritte zueinander.

□

Für die hier verfolgten Zwecke ist das bereits angesprochene Charakteristikum des zweiten Präfixes besonders wichtig, nämlich daß die errechnete Substitution (höchstens) eine Variablenumbenennung ist.
Um nun die einzelnen Schritte zusammensetzen zu können, werden schließlich noch die folgenden zwei Lemmata benötigt:

Lemma 3.3.8.
In einem NN-Schritt $t_1 \rightsquigarrow_{p,l\to r,\sigma} t_2$ gilt

$$\sigma = \big(mgu(l, \psi(t_1)|_p)\big)[\mathcal{V}ar(t_1)]$$

mit $\psi = \varphi_n \circ \cdots \circ \varphi_1$, wobei

$$\begin{aligned}
\varphi_1 &= leftmgu^{o_1}(t_1, l_1),\\
\varphi_2 &= leftmgu^{o_2}(\varphi_1(t_1)|_{o_1}, l_2),\\
&\vdots\\
\varphi_{n-1} &= leftmgu^{o_{n-1}}(\varphi_{n-2}(t_1)|_{o_1\cdots o_{n-2}}, l_{n-1}),\\
\varphi_n &= leftmgu^{o_n}(\varphi_{n-1}(t_1)|_{o_1\cdots o_{n-1}}, l_n)
\end{aligned}$$

ist und dabei

1. $n = \Big|\big\{q \big| q \prec p \wedge t_1|_q \in \mathcal{F}\big\}\Big|$, *d.h. n die Anzahl der Funktionssymbole über p ist,*
2. $o_1 \prec o_2 \prec \ldots \prec o_n$ *die Positionen dieser Funktionssymbole sind und*
3. $l_1, \ \ldots, l_n$ *Instanzen der Muster $\pi_1, \ \ldots, \pi_n$ derjenigen von der λ'-Aufruffolge erreichten Knoten sind, in denen ein Fall $R_4^{\lambda'}$ auftrat.*

Beweis
erfolgt identisch zum Beweis von Satz 3.2.14 durch vollständige Induktion über die Länge der λ'-Aufruffolge. Es ist $mgu(t_1|_{o_1\cdots o_i}, \pi_i) = leftmgu^{o_i}(t_1|_{o_1\cdots o_{i-1}}, l_i)$ für alle Instanzen l_i von π_i nach Definition von λ'.

□

Lemma 3.3.9. *Ersetzt man in der dritten Feststellung von Lemma 3.3.8 die l_i durch π_i für $0 \leq i < n$, wobei die π_i die Muster derjenigen Knoten sind, in denen ein Fall $R_4^{\lambda'}$ auftritt, so kann man auch die $leftmgu^{o_{i+1}}(t_1|_{o_1\cdots o_i}, l_{i+1})$*

durch $mgu(t_1|_{o_1 \cdots o_{i+1}}, \pi_{i+1})$ *ersetzen und erhält das gleiche Ergebnis für die Substitutionen eingeschränkt auf* $\mathcal{V}ar(t_1)$. *Wesentlich ist, daß die linken Regelseiten und damit die Muster linear sind.*

Beweis
erfolgt identisch zum Beweis von Lemma 3.3.8. Lemma 3.3.8 stellt eine Verfeinerung von Lemma 3.3.9 dar.

□

Im nächsten Satz wird nun eine NN-Ableitung durch eine OINC-Ableitung simuliert. Da die Formulierung unübersichtlich ist, folgt zunächst die "Quintessenz" der Aussage:

Satz 3.3.10.
Wenn es eine NN-Ableitung

$$t_1 \rightsquigarrow_{p_1, l_1 \to r_1, \sigma_1} t_2 \rightsquigarrow \cdots \rightsquigarrow t_{n-1} \rightsquigarrow_{p_{n-1}, l_{n-1} \to r_{n-1}, \sigma_{n-1}} t_n$$

gibt, dann gibt es eine OINC-Ableitung

$$\begin{array}{rcl} t_1 \approx \mathit{true} & \rightsquigarrow^* & \sigma'_1\big(r_1 \approx u_1, E_1\big) \\ & \rightsquigarrow^* & \sigma'_2 \circ \sigma'_1\big(r_2 \approx u_2, E_2\big) \\ & \vdots & \\ & \rightsquigarrow^* & \sigma'_{n-1} \circ \cdots \circ \sigma'_1\big(r_{n-1} \approx u_{n-1}, E_{n-1}\big) \end{array}$$

mit Termen $u_1, \ldots, u_{n-1}$*, Gleichungsfolgen* $E_1, \ldots, E_{n-1}$ *und* $\sigma_i = \sigma'_i$ *für* $1 \leq i < n$.
Dabei ist die Anzahl der [on]-Inferenzen gleich der Anzahl der NN-Schritte.

Beweis
Vgl. den Beweis von Satz 3.3.11

□

Satz 3.3.11.
Wenn es eine NN-Sequenz der Form

$$t_1 \rightsquigarrow_{p_1, L_1 \to R_1, \sigma_1} t_2 \rightsquigarrow \ldots \rightsquigarrow_{p_{n-2}, L_{n-2} \to R_{n-2}, \sigma_{n-2}} t_{n-1} \rightsquigarrow_{p_{n-1}, L_{n-1} \to R_{n-1}, \sigma_{n-1}} t_n$$

gibt, dann gibt es für eine OINC-geeignete Gleichung $t_1 \approx s$ *mit* $s \notin \mathcal{V}ar$ *eine OINC-Sequenz der Form*

$$
\begin{aligned}
t_1 \approx s \rightsquigarrow^* \sigma_1(R_1 &\approx l_{i_1-1}|_{q_{i_1-1}\cdot w_{i_1-1}}, t_2|_{v_{i_1-1}\cdot(q_{i_1-1}+1)} \approx l_{i_1-1}|_{q_{i_1-1}+1}, \ \dots, \\
&\quad t_2|_{v_{i_1-1}\cdot n_{i_1-1}} \approx l_{i_1-1}|_{n_{i_1-1}}, \\
&r_{i_1-1} \approx l_{i_1-2}|_{q_{i_1-2}\cdot w_{i_1-2}}, t_2|_{v_{i_1-2}\cdot(q_{i_1-2}+1)} \approx l_{i_1-2}|_{q_{i_1-2}+1}, \ \dots, \\
&\quad t_2|_{v_{i_1-2}\cdot n_{i_1-2}} \approx l_{i_1-2}|_{n_{i_1-2}}, \\
&\vdots \\
&r_2 \approx l_1|_{q_1\cdot w_1}, t_2|_{v_1\cdot(q_1+1)} \approx l_1|_{q_1+1}, \ \dots, t_2|_{v_1\cdot n_1} \approx l_1|_{n_1}, \\
&r_1 \approx s) \\
\rightsquigarrow^* \sigma_2 \circ \sigma_1(R_2 &\approx l_{i_2-1}|_{q_{i_2-1}\cdot w_{i_2-1}}, t_3|_{v_{i_2-1}\cdot(q_{i_2-1}+1)} \approx l_{i_2-1}|_{q_{i_2-1}+1}, \ \dots, \\
&\quad t_3|_{v_{i_2-1}\cdot n_{i_2-1}} \approx l_{i_2-1}|_{n_{i_2-1}}, \\
&r_{i_2-1} \approx l_{i_2-2}|_{q_{i_2-2}\cdot w_{i_2-2}}, t_3|_{v_{i_2-2}\cdot(q_{i_2-2}+1)} \approx l_{i_2-2}|_{q_{i_2-2}+1}, \ \dots, \\
&\quad t_3|_{v_{i_2-2}\cdot n_{i_2-2}} \approx l_{i_2-2}|_{n_{i_2-2}}, \\
&\vdots \\
&r_2 \approx l_1|_{q_1\cdot w_1}, t_3|_{v_1\cdot(q_1+1)} \approx l_1|_{q_1+1}, \ \dots, t_3|_{v_1\cdot n_1} \approx l_1|_{n_1}, \\
&r_1 \approx s) \\
\vdots \\
\rightsquigarrow^* \sigma_{n-1} \circ \dots \circ \sigma_1(R_{n-1} &\approx l_{i_{n-1}-1}|_{q_{i_{n-1}-1}\cdot w_{i_{n-1}-1}}, \\
&t_n|_{v_{i_{n-1}-1}\cdot(q_{i_{n-1}-1}+1)} \approx l_{i_{n-1}-1}|_{q_{i_{n-1}-1}+1}, \ \dots, \\
&\quad t_n|_{v_{i_{n-1}-1}\cdot n_{i_{n-1}-1}} \approx l_{i_{n-1}-1}|_{n_{i_{n-1}-1}}, \\
&r_{i_{n-1}-1} \approx l_{i_{n-1}-2}|_{q_{i_{n-1}-2}\cdot w_{i_{n-1}-2}}, \\
&\quad t_n|_{v_{i_{n-1}-2}\cdot(q_{i_{n-1}-2}+1)} \approx l_{i_{n-1}-2}|_{q_{i_{n-1}-2}+1}, \ \dots, \\
&\quad t_n|_{v_{i_{n-1}-2}\cdot n_{i_{n-1}-2}} \approx l_{i_{n-1}-2}|_{n_{i_{n-1}-2}}, \\
&\vdots \\
&r_2 \approx l_1|_{q_1\cdot w_1}, t_n|_{v_1\cdot(q_1+1)} \approx l_1|_{q_1+1}, \ \dots, t_n|_{v_1\cdot n_1} \approx l_1|_{n_1}, \\
&r_1 \approx s),
\end{aligned}
$$

wobei

1. $i_j = \Big|\{p|p \preceq p_j \wedge t_j|_p \in \mathcal{F}\}\Big|$ *für* $1 \leq j < n$ *die Anzahl der Funktionssymbole über* p_j *in* t_j *(*$Head(t_j|_{p_j})$ *eingeschlossen) bezeichnet,*

2. v_k *für* $1 \leq k < i_j$ *und* $1 \leq j < n$ *die Positionen aller über* p_j *gelegenen Funktionssymbole über* p_j *in* t_j *sind (von oben nach unten geordnet, d.h.* $Head(t_j|_{v_k}) \in \mathcal{F}$ *und* $v_1 \prec v_2 \prec \dots \prec v_{i_j-1}$*),* $v_{i_j} = p_j$ *und* $v_0 = \varepsilon$ *ist,*

3. $v_k \cdot q_k$ *für* $1 \leq k < N$ *die direkt unter* v_k *und über* v_{k+1} *liegende Position ist und* w_k *den Pfad bis* v_{k+1} *vervollständigt, d.h.* $v_{k+1} = v_k \cdot q_k \cdot w_k$ *ist (dabei sei* $q_0 = \varepsilon$*,* $v_0 \cdot q_0 = \varepsilon$ *und im Fall* $v_k = v_{k+1} : v_k \cdot q_k = v_k = v_{k+1}$*),*

4. $p_j = v_{i_j} = v_{i_j-1} \cdot q_{i_j-1} \cdot w_{i_j-1}$ *für* $1 \leq j < n$ *mit* q_{i_j-1} *atomar (oder* ε*) ist,*

5. n_k *für* $1 \leq k < i_j$ *die Stelligkeit des nächsten über* v_{k+1} *gelegenen Funktionssymbols bezeichnet, d.h.* $n_k = Arity\big(Head(t_j|_{v_k})\big)$*,*

6. $l_k \to r_k$ *für* $1 \leq k < i_j$ *und* $1 \leq j < n$ *diejenige Regel bezeichnet, mit der*

ein [on]-Schritt an $t_m|_{v_{i_j}}$ für ein $m < j$ durchgeführt wurde und $l_0 = s$ ist (dann bestehen die Gleichungsmengen nur aus der ersten Gleichung) und

7. *die Anzahl der [on]-Inferenzen gleich der Anzahl der NN-Schritte ist.*

Bemerkung 3.3.12.
Die Substitutionen σ_i in beiden Ableitungen sind nur eingeschränkt auf $\mathcal{V}ar(t_i)$ für $1 \leq i < n$ modulo Variablenumbenennung identisch. Das wird im folgenden der Einfachheit halber ignoriert.

Beweis
durch vollständige Induktion über die Länge n-1 der NN-Sequenz.

Induktionsverankerung: $t_1 \rightsquigarrow_{p,L\to R,\sigma} t_2$
Der Beweis ähnelt demjenigen von Satz 3.3.4; er wird hier mit modifizierter Notation vereinfacht wiederaufgenommen, da im Induktionsschritt Rückgriffe auf ihn notwendig sind.
Nach Lemma 3.2.13 entspricht eine [on]-Inferenz in OINC dem Auftreten eines Falls $R_4^{\lambda'}$ in NN bei der Errechnung der Substitution. Es werden nun im folgenden immer Regeln verwendet, deren linke Seite Instanz des Musters desjenigen Knotens eines DT ist, der bei dem nächsten Auftreten eines Falls $R_4^{\lambda'}$ erreicht wurde. Diese Regel muß natürlich nicht eindeutig festgelegt sein. Wenn es mehrere linke Regelseiten gibt, die Instanz des Musters sind, müssen alle ausprobiert werden. Insbesondere wird also auch die in bezug auf NN "richtige" Regel verwendet.
N sei die Anzahl der Auftreten des Falls $R_4^{\lambda'}$ an den Verzweigungspositionen $o_1, \ldots, o_N$ und damit gleich der Anzahl der über $t_1|_p$ liegenden Funktionssymbole ($Head(t_1|_p)$ nicht eingeschlossen), also ist $N + 1 = i_1$ für das i_1 aus der Formulierung des Satzes. Es sei $o_1 \cdots o_{k+1} = o_1 \cdots o_k \cdot q_k \cdot w_k$ mit $1 < k \leq N$ für ein atomares q_k. Demzufolge entspricht $o_1 \cdots o_k$ für $1 \leq k < N$ den v_{k+1} aus der Formulierung des Satzes, und es ist insbesondere $p = o_1 \cdots o_N$ (die Indexverschiebung ergibt sich aus der (formalen) Tatsache, daß der erste λ'-Aufruf an der Wurzelposition in dieser Notation kein Auftreten des Falls $R_4^{\lambda'}$ darstellt).
Dann gibt es eine OINC-Inferenz

$$
\begin{aligned}
t_1 \approx s \rightsquigarrow_{[on]} & \; t_1|_1 \approx l_1|_1,\ \dots, t_1|_{n_1} \approx l_1|_{n_1}, r_1 \approx s \\
\rightsquigarrow^*_{[d,v]} & \; \varphi_1(t_1|_{o_1} \approx l_1|_{o_1}, \overbrace{t_1|_{q_1+1} \approx l_1|_{q_1+1},\ \dots, t_1|_{n_1} \approx l_1|_{n_1}, r_1 \approx s}^{E_1}) \\
\rightsquigarrow_{[on]} & \; \varphi_1(t_1|_{o_1\cdot 1} \approx l_2|_1,\ \dots, t_1|_{o_1\cdot n_2} \approx l_2|_{n_2}, r_2 \approx l_1|_{o_1}, E_1) \\
\rightsquigarrow^*_{[d,v]} & \; \varphi_2 \circ \varphi_1(t_1|_{o_1\cdot o_2} \approx l_2|_{o_2}, \\
& \quad \overbrace{t_1|_{o_1\cdot(q_2+1)} \approx l_2|_{q_2+1},\ \dots, t_1|_{o_1\cdot n_2} \approx l_2|_{n_2}, r_2 \approx l_1|_{o_1}}^{E_2}, E_1) \\
\vdots & \\
\rightsquigarrow^*_{[d,v]} & \; \varphi_N \circ \dots \circ \varphi_1(t_1|_{o_1\cdots o_N} \approx l_N|_{o_N}, \\
& \quad \left.\begin{array}{l} t_1|_{o_1\cdots o_{N-1}\cdot(q_N+1)} \approx l_N|_{q_N+1},\ \dots, t_1|_{o_1\cdots o_{N-1}\cdot n_N} \approx l_N|_{n_N}, \\ r_N \approx l_{N-1}|_{o_{N-1}}, \end{array}\right\} E_N \\
& \quad E_{N-1},\ \dots, E_1) \\
\rightsquigarrow_{[on]} & \; \varphi_N \circ \dots \circ \varphi_1(t_1|_{p\cdot 1} \approx L|_1,\ \dots, t_1|_{p\cdot\tilde{n}} \approx L|_{\tilde{n}}, R \approx l_N|_{o_N}, \\
& \quad E_N,\ \dots, E_1) \\
\rightsquigarrow^*_{[d,v]} & \; \psi \circ \varphi_N \circ \dots \circ \varphi_1(R \approx l_N|_{o_N}, E_N,\ \dots, E_1)
\end{aligned}
$$

mit

1. $E_i = t_1|_{o_1\cdots o_{i-1}\cdot(q_i+1)} \approx l_i|_{q_i+1},\ \dots, t_1|_{o_1\cdots o_{i-1}\cdot n_i} \approx l_i|_{n_i}, r_i \approx l_{i-1}|_{o_{i-1}}$ für $1 \leq i \leq N$ mit $o_0 = \varepsilon$,
2. $\tilde{n} = Arity(Head(t_1|_p))$,
3. $n_i = Arity(Head(t_1|_{o_1\cdots o_{i-1}}))$ für $1 \leq i \leq N$ mit $o_0 = \varepsilon$ und
4. $\varphi_1 = leftmgu^{o_1}(t_1, l_1),\ \varphi_2 = leftmgu^{o_2}(\varphi_1(t_1)|_{o_1}, l_2),\ \dots,$ $\varphi_N = leftmgu^{o_N}(\varphi_{N-1} \circ \dots \circ \varphi_1(t_1)|_{o_1\cdots o_{N-1}}, l_N)$, $\psi = mgu(\varphi_N \circ \dots \circ \varphi_1(t_1|_p), L)$.

Der Fall $N = 0$ wird von der angegebenen Ableitung ebenfalls erfaßt, da nach Definition $l_0 = s = t_1$ gilt. Dieser Fall entspricht dem Narrowing an der Wurzelposition.

Wenn $\pi_1,\ \dots, \pi_N$ die Muster derjenigen Knoten mit Verzweigungsposition $o_1,\ \dots, o_N$ sind, in denen ein Fall $R_4^{\lambda'}$ auftritt, dann ist nach Lemma 3.3.9 $\sigma = \big(mgu(\psi'(t_1)|_p, L)\big)[\mathcal{V}ar(t_1)]$ für $\psi' = \varphi'_N \circ \dots \circ \varphi'_1$ mit

$$
\begin{aligned}
\varphi'_1 &= mgu(t_1, \pi_1), \\
\varphi'_2 &= mgu(\varphi'_1(t_1)|_{o_1}, \pi_2), \\
&\vdots \\
\varphi'_{N-1} &= mgu(\varphi'_{N-2}(t_1)|_{o_1\cdots o_{N-2}}, \pi_{N-1}), \\
\varphi'_N &= mgu(\varphi'_{N-1}(t_1)|_{o_1\cdots o_{N-1}}, \pi_N).
\end{aligned}
$$

Nach Konstruktion sind $l_1,\ \dots, l_N$ Instanzen dieser Muster, und damit existieren die $leftmgu^{o_i}(t_1|_{o_1\cdots o_{i-1}}, \varphi_{i-1}(l_i)) = \varphi_i = mgu(t_1|_{o_1\cdots o_i}, \varphi'_{i-1}(\pi_i))$ mit $\varphi_0 = \varphi'_0 = \emptyset$ bei Einschränkung auf $\mathcal{V}ar(t)$, denn nach Definition der LR-DT sind die Teilterme der Muster π_i rechts von o_i Variablen, und die (jeweils frischen) Regeln sind linkslinear. Lemma 3.2.1 und Lemma 3.2.3 liefern dann die

Existenz der [d,v]-Sequenzen zwischen den [on]-Schritten; damit ist die 4. Eigenschaft über Existenz und Aussehen der Substitutionen nachgewiesen (Lemma 3.3.8; es ist $\varphi_i(t_{i-1}) = \varphi_i \circ \cdots \varphi_1(t_{i-1})$ für $1 \leq i \leq N$ wegen der Linearität der linken Regelseiten und Idempotenz der Substitutionen). Nach Konstruktion ist $Head(t_1|_{o_1 \cdots o_i}) \in \mathcal{F}$ für $1 \leq i \leq N$, und damit ist die Existenz der gesamten angegebenen OINC-Ableitung gezeigt. Mit $o_i =: q_i \cdot w_i$ und $o_1 \cdots o_i =: v_{i+1}$ für $1 \leq i < N$ ergibt sich auch die Übereinstimmung der Positionen mit den im Satz formulierten.
$\sigma = \psi \circ \varphi_N \circ \cdots \circ \varphi_1[\mathcal{V}ar(t_1)]$ folgt jetzt direkt durch Einsetzen, und die sechs Bedingungen aus der Formulierung des Satzes sind durch die Konstruktion erfüllt. Zu zeigen bleibt, daß man alle t_1 in der inferierten Gleichungsfolge durch t_2 ersetzen kann. Das ist der Fall, da nach Konstruktion in den E_i für $1 \leq i \leq N$ nur Teilterme stehen, die rechts von p in t_1 liegen. Wegen $t_2 = \sigma(t_1[r]_p)$ ist $t_2|_q = \sigma(t_1)|_q$ für $q \in Right(t_1, p)$; damit ist die Umbenennung unproblematisch.

Induktionsschritt: $t_1 \rightsquigarrow^* t_{n-1} \rightsquigarrow_{p_{n-1}, L_{n-1} \to R_{n-1}, \sigma_{n-1}} t_n \rightsquigarrow_{p_n, L_n \to R_n, \sigma_n} t_{n+1}$
Der Beweis wird durch Fallunterscheidung über die relative Position von p_n in bezug auf p_{n-1} geführt.

1. Fall: $p_n \prec p_{n-1}$.
Dann gibt es kein p mit $p_n \prec p \prec p_{n-1}$ und $Head(t_n|_p) \in \mathcal{F}$, weil die betrachteten Regeln induktiv-sequentiell sind (wenn p_n erreicht wurde, muß sofort vorher auf das direkt über p_n liegende Funktionssymbol eine [on]-Inferenz angewendet worden sein. Läge dieses zwischen p_{n-1} und p_n, dann müßte es ein Muster in einem DT geben, das Funktionssymbole an mindestens einer anderen Stelle als der Wurzelposition enthielte, was in induktiv-sequentiellen TES nicht der Fall sein kann).
Nach Lemma 3.3.8 ist $\sigma_n = mgu(\psi(t_n)|_{p_n}, L_n)$ mit $\psi = \varphi_N \circ \cdots \circ \varphi_1$ für $\varphi_i = leftmgu^{o_i}(\varphi_{i-1}(t_n)|_{o_1 \cdots o_{i-1}}, l_i)$ mit $1 \leq i \leq N$ und $\varphi_0 = \emptyset$, wobei N die Anzahl der Funktionssymbole über $t_n|_{p_n}$ bezeichnet und $o_1, \ldots, o_N$ die Positionen dieser Symbole sind.
Aus $p_n \prec p_{n-1}$ folgt mit Lemma 3.3.7 wegen $p_n = Pos_{\top}(p_n, p_{n-1}, t_n, t_{n-1})$: $\sigma_n = mgu(t_n|_{p_n}, L_n)$, denn die λ'-Aufruffolgen sind in bezug auf die besuchten Knoten identisch, und damit ist ψ (höchstens) eine Variablenumbenennung.
Die Induktionsvoraussetzung liefert mit $R_{n-1} = r_{N+1}$ eine OINC-Ableitung mit Resultat

$$\begin{aligned}\sigma_{n-1} \circ \cdots \circ \sigma_1(R_{n-1} \approx l_N|_{o_N},\\ & t_n|_{o_1 \cdots o_{N-1} \cdot (q_N+1)} \approx l_N|_{q_N+1}, \ \ldots, t_n|_{o_1 \cdots o_{N-1} \cdot n_N} \approx l_N|_{n_N},\\ & r_N \approx l_{N-1}|_{o_{N-1}}, E),\end{aligned}$$

wobei N, o_i und q_i für $1 \leq i \leq N$ wie in der Induktionsverankerung definiert sind. Also ist auch $o_1 \cdots o_N = p_{n-1}$ und $o_1 \cdots o_{N-1} = p_n$ (zwischen p_{n-1} und p_n liegen keine Funktionssymbole). Nach obiger Feststellung sind $t_n|_{p_n}$ und L_n unifizierbar mittels σ_n. Es kann weiterhin $l_N = L_n$ angenommen werden, da nach Voraussetzung bei der Auswahl der Regeln für die [on]-Schritte immer *alle* Regeln angewendet werden, deren linke Seite Instanz des Musters des nächsten erreichten Knotens mit Auftreten des Falls $R_4^{\lambda'}$ ist: Für den Schritt $t_{n-1} \rightsquigarrow t_n$ fand das letzte Auftreten von $R_4^{\lambda'}$ an Position p_{n-1} in t_{n-1} statt; es wurde also eine Regel $l_N \to r_N$ gewählt, für die der $leftmgu^{o_N}(t_{n-1}|_{p_n}, l_N)$ existierte. L_n erfüllt offensichtlich dieses Kriterium wegen $p_n \prec p_{n-1}$ und der Unifizierbarkeit von L_n und $t_n|_{p_n}$. Also gibt es eine OINC-Sequenz, die die Regel $L_n \to R_n$ bei Erreichen der Position p_n verwendet hat.

Aus der Unifizierbarkeit von L_n und $t_n|_{p_n} = \sigma_n(t_{n-1}[R_{n-1}]_{p_{n-1}})|_{p_n}$ folgt auch die Unifizierbarkeit von R_{n-1} und $l_N|_{o_N}$ und der restlichen Terme unterhalb von o_{N-1} mittels einer Substitution

$$leftmgu^{o_N}(t_n|_{p_n}, L_n) \circ mgu(R_{n-1}, L_n|_{o_N}) \circ rightmgu^{o_N}(t_n|_{p_n}, L_n);$$

dabei ist die Reihenfolge der Substitutionsanwendung irrelevant (linkslineare Regeln).
Nun ist

$$\begin{aligned}\sigma_n &= mgu(t_n|_{p_n}, L_n)\\ &= rightmgu^{o_N}(t_n|_{p_n}, L_n) \circ mgu(t_n|_{p_n \cdot o_N}, L_n|_{o_N}) \circ leftmgu^{o_N}(t_n|_{p_n}, L_n).\end{aligned}$$

Dabei ist $leftmgu^{o_N}(t_n|_{p_n}, L_n) \restriction_{\mathcal{V}ar(t_n)}$ höchstens eine Variablenumbenennung, weil die im Schritt $t_{n-1} \rightsquigarrow t_n$ und $t_n \rightsquigarrow t_{n+1}$ auf das Auftreten des Falls $R_4^{\lambda'}$ an Position p_n folgenden Verzweigungen im Baum für $Head(t_{n-1}|_{p_n})$ bis zur Position p_{n-1} (d.h. Verzweigungsposition o_N wegen $p_{n-1} = p_n \cdot o_N$) identisch sind: Dazu zeigt man zunächst leicht durch Einsetzen, daß L_n Instanz desjenigen Musters sein muß an dem im ersten Schritt an der Position p_{n-1} ein Fall $R_4^{\lambda'}$ auftrat. Bei Verwendung von LR-DTs gilt nun: Wenn ein Argument einer linken Regelseite mit demselben Argument einer anderen linken Regelseite für dieselbe Funktion unifizierbar ist, dann sind diese Argumente identisch bis auf Variablenumbenennung, oder es gibt ein weiter links stehendes Argument, das mit dem entsprechenden Argument der anderen linken Regelseite nicht unifizierbar ist. Anders gesagt: Wenn der $leftmgu$ zweier Muster an einer Position o existiert, dann ist er bei Verwendung von LR-DTs höchstens eine Variablenumbenennung. Damit legt der Schritt $t_{n-1} \rightsquigarrow t_n$ das Verhalten von λ' im Schritt $t_n \rightsquigarrow t_{n+1}$ an diesen Positionen links von o_N fest.
Da σ_n existiert, existieren insbesondere die Komponenten $mgu(t_n|_{p_n \cdot o_N}, L_n|_{o_N})$ und $rightmgu^{o_N}(t_n|_{p_n}, L_n)$. Diese können nun gemäß Lemma 3.2.1 und Lemma 3.2.3 mittels der OINC-Sequenz

$$\ldots \rightsquigarrow^*_{[d,v]} \sigma_n \circ \cdots \circ \sigma_1(r_N \approx l_{N-1}|_{o_{N-1}}, E)$$

berechnet werden.
Wegen $p_n \prec p_{n-1}$ ändert sich die Gleichungsfolge E während der Rechnung nur in bezug auf Variablenbindungen mittels σ_n; wenn E nach Induktionsvoraussetzung die in der Formulierung des Satzes geforderte Gestalt hat, hat auch die zuletzt inferierte Folge diese Gestalt. Also sind alle sechs Bedingungen erfüllt.
Schließlich kann in E jedes t_n durch ein t_{n+1} mit derselben Argumentation wie in der Induktionsverankerung umbenannt werden.

2.Fall: $p_n \succeq p_{n-1}$
Dann wird das Narrowing in der zuletzt ersetzten rechten Regelseite durchgeführt. Der Beweis erfolgt analog zum Beweis der Induktionsverankerung; im Fall $p_n \succ p_{n-1}$ muß man zunächst mit einer [d,v]-Sequenz bis zum ersten Funktionssymbol unter p_{n-1} absteigen. Dann ist die Konstruktion der Ableitung identisch zu der in der Induktionsverankerung angegebenen.

3.Fall: $p_{n-1} \in Left(p_n)$
Es sei $p_{n-1} = o_1 \cdots o_N$ und $p_n = \overline{o}_1 \cdots \overline{o}_{\overline{N}}$, wobei $o_1 \prec \ldots \prec o_N$ und $\overline{o}_1 \prec \ldots \prec \overline{o}_{\overline{N}}$ die Positionen der Funktionssymbole über p_{n-1} in t_{n-1} und über p_n in t_n darstellen (p_{n-1} bzw. p_n eingeschlossen). $\tilde{o} = o_1 \cdots o_j = Pos_{\top}(p_{n-1}, p_n, t_{n-1}, t_n)$ bezeichne das längste gemeinsame Präfix von p_{n-1} und p_n.
Dann muß $j = N - 1$ wegen der Betrachtung induktiv-sequentieller TES sein, d.h. zwischen $\tilde{o}$ und p_{n-1} stehen keine Funktionssymbole (vgl. die Argumentation im 1. Fall): Angenommen, das Gegenteil ist der Fall. Dann gibt es für $k > 0$ Positionen q_i mit $\tilde{o} \prec q_1 \prec \cdots \prec q_k \prec p_{n-1}$ mit $t_{n-1}|_{q_i} \in \mathcal{F}$ für $1 \leq i \leq k$. Im Schritt $t_{n-1} \rightsquigarrow t_n$ fand dann wegen $\tilde{o} \prec q_1 \prec p_{n-1}$ an Position $\tilde{o}$ ein Auftreten des Falls $R_4^{\lambda'}$ statt (in einem Knoten mit Muster π), d.h. $t_{n-1}|_{\tilde{o}}$ und π sind nicht unifizierbar. Für $q_1 = \tilde{o} \cdot u_1$ sind dann insbesondere $\pi|_{u_1}$ und $t_{n-1}|_{q_1}$ nicht unifizierbar (sonst würde der Narrowingschritt wegen der Verwendung von LR-DTs nicht an Position p_{n-1} stattfinden, sondern links davon). Wegen $t_{n-1}|_{q_1} \in \mathcal{F}$ und $q_1 \prec p_{n-1}$ folgt dann wegen der Verwendung von induktiv-sequentiellen TES, die außer an der Wurzelposition keine Funktionssymbole enthalten dürfen, daß auch $\pi|_{u_1}$ und $t_n|_{q_1} = \sigma_{n-1}(t_{n-1}[R_{n-1}]_{p_{n-1}})|_{q_1}$ nicht unifizierbar sein können. Damit müßte der nächste Narrowing-Schritt $t_n \rightsquigarrow t_{n+1}$ an einer Position $q' \succeq \tilde{o} \cdot u_1$ stattfinden, was nicht der Fall ist, weil er an $p_n \in Right(t_{n-1}, p_{n-1})$ stattfindet.
Damit kann es kein solches q_1 geben; zwischen $\tilde{o}$ und p_{n-1} stehen keine Funktionssymbole.

Also ist $\tilde{o} = o_1 \cdots o_{N-1}$, $p_{n-1} = \tilde{o} \cdot o_N$ und $p_n = \tilde{o} \cdot \overline{o}_N \cdot \overline{o}_{N+1} \cdots \overline{o}_{\overline{N}}$.
Die Induktionsvoraussetzung liefert eine OINC-Ableitung mit Resultat

$$\begin{aligned}\sigma_{n-1} \circ \cdots \circ \sigma_1(&R_{n-1} \approx l_N|_{o_N},\\ &t_n|_{\tilde{o}\cdot(q_{N-1}+1)} \approx l_N|_{q_{N-1}+1},\ \ldots, t_n|_{\tilde{o}\cdot n_N} \approx l_N|_{n_N},\\ &r_N \approx l_{N-1}|_{o_{N-1}}, E),\end{aligned}$$

wobei N die Anzahl der Funktionsymbole über p_{n-1} bezeichnet, $o_N = q_{N-1} \cdot w_{N-1}$ für ein atomares q_{N-1}, $n_N = Arity(Head(t_n|_{\tilde{o}}))$ und $l_N \to r_N$ diejenige Regel ist, die an der Position $\tilde{o}$ Anwendung fand.
Die von λ' berechnete Substitution ist nach Lemma 3.3.7 bis zur Position $\tilde{o}$ höchstens eine Variablenumbenennung. Der nächste erreichte DT-Knoten, in dem ein Fall $R_4^{\lambda'}$ auftritt, hat nach Voraussetzung die Verzweigunsgposition $\overline{o}_N$. Es sei $\overline{o}_N = \overline{q}_{N-1} \cdot \overline{w}_{N-1}$ für ein atomares $\overline{q}_{N-1}$. Da sich q_{N-1} und $\overline{q}_{N-1}$ auf einer Ebene im Term befinden, sei $q_{N-1} + k = \overline{q}_{N-1}$ für ein $k > 0$.
Dann ist $\sigma_n \geq leftmgu^{\overline{o}_N}(t_n|_{\tilde{o}}, l_N)$ (auf die dem $leftmgu$ nachzuschaltende Substitution ψ wird später eingegangen. Sie hat natürlich die in Lemma 3.3.9 angegebene Form).

Nun gilt

$$\begin{aligned} leftmgu^{\overline{o}_N}(t_n|_{\tilde{o}}, l_N) = \quad & mgu(t_n|_{\tilde{o}\cdot(q_{N-1}+k-1)}, l_N|_{q_{N-1}+k-1}) \\ \circ \quad & mgu(t_n|_{\tilde{o}\cdot(q_{N-1}+k-2)}, l_N|_{q_{N-1}+k-2}) \\ \circ \quad & \cdots \\ \circ \quad & mgu(t_n|_{\tilde{o}\cdot(q_{N-1}+1)}, l_N|_{q_{N-1}+1}) \\ \circ \quad & leftmgu^{q_{N-1}}(t_n|_{\tilde{o}}, l_N), \end{aligned}$$

weil l_N linear ist.
Dabei ist $leftmgu^{q_{N-1}}(t_n|_{\tilde{o}}, l_N) \restriction_{\mathcal{V}ar(t_n)}$ höchstens eine Variablenumbenennung, weil die auf den an Position $\tilde{o}$ auftretenden Fall $R_4^{\lambda'}$ folgenden Verzweigungen im definierenden Baum für $Head(t_n|_{\tilde{o}})$ bis zur Verzweigungsposition o_N wegen der Verwendung von LR-DTs im Schritt $t_{n-1} \rightsquigarrow t_n$ und $t_n \rightsquigarrow t_{n+1}$ identisch sind (vgl. die Argumentation im ersten Fall).
Nach Voraussetzung wurde auch von OINC die "richtige" Regel $l_N \to r_N$ gewählt (bzw. es gibt eine Ableitung, in der diese "richtige" Regel gewählt wurde). Da der $leftmgu^{\overline{o}_N}(t_n|_{\tilde{o}}, l_N) =: \eta$ existiert (sonst könnte kein NN-Schritt mit dieser Regel durchgeführt werden), gibt es für $\xi = \sigma_{n-1} \circ \cdots \circ \sigma_1$ wegen $t_n = \sigma_{n-1}(t_{n-1}[R_{n-1}]_{p_{n-1}})$ mit Lemma 3.2.1 die OINC-Sequenz

$$\begin{aligned} \ldots \rightsquigarrow^* \quad & \eta' \circ \xi(t_n|_{\tilde{o}\cdot(q_{N-1}+1)} \approx l_N|_{q_{N-1}+1}, \ \ldots, t_n|_{\tilde{o}\cdot n_N} \approx l_N|_{n_N}, \\ & \quad r_N \approx l_{N-1}|_{o_{N-1}}, E) \\ \rightsquigarrow^* \quad & \chi_1 \circ \eta' \circ \xi(t_n|_{\tilde{o}\cdot(q_{N-1}+2)} \approx l_N|_{q_{N-1}+2}, \ \ldots, t_n|_{\tilde{o}\cdot n_N} \approx l_N|_{n_N}, \\ & \quad r_N \approx l_{N-1}|_{o_{N-1}}, E) \\ \rightsquigarrow^* \quad & \chi_{k-1} \circ \cdots \circ \chi_1 \circ \eta' \circ \xi(t_n|_{\tilde{o}\cdot\overline{q}_{N-1}} \approx l_N|_{\overline{q}_{N-1}}, \ \ldots, t_n|_{\tilde{o}\cdot n_N} \approx l_N|_{n_N}, \\ & \quad r_N \approx l_{N-1}|_{o_{N-1}}, E) \\ \rightsquigarrow^*_{[d]} \quad & \chi_{k-1} \circ \cdots \circ \chi_1 \circ \eta' \circ \xi(t_n|_{\tilde{o}\cdot\overline{o}_N} \approx l_N|_{\overline{o}_N}, \ \ldots, t_n|_{\tilde{o}\cdot n_N} \approx l_N|_{n_N}, \\ & \quad r_N \approx l_{N-1}|_{o_{N-1}}, E) \end{aligned}$$

mit $\eta = \eta'$ und $\chi_i = mgu(t_n|_{\tilde{o}\cdot(q_{N-1}+i)}, l_N|_{q_{N-1}+i})$ für $1 \leq i < k$. Einsetzen liefert $leftmgu^{\overline{o}_N}(t_n|_{\tilde{o}}, l_N) = \chi_{k-1} \circ \cdots \circ \chi_1 \circ \eta'$ wegen $\xi(t_n) = t_n$.

Nun folgen $\overline{N} - N$ Narrowingschritte (mit den dazwischenliegenden Dekompositionen und Variablenbindungen) wie im Beweis der Induktionsvoraussetzung. Die dabei berechnete Substitution ist nach Lemma 3.3.9 die weiter oben angesprochene Substitution ψ. Dann kann wie dort das Narrowing an Position p_n mit der Regel $L_n \to R_n$ durchgeführt werden (identischer Beweis wie in der Induktionsvoraussetzung).
Wie im ersten Fall ergibt sich, daß die Umbenennung der t_n durch t_{n+1} unkritisch ist. Aus der angegebenen Konstruktion ergibt sich das in der Behauptung angegebene Aussehen der zuletzt inferierten Gleichungsmenge.

4. Fall: $p_{n+1} \in Left(p_n, t_n)$
Dieser Fall kann wegen der Verwendung von LR-DTs nicht auftreten. Die Regeln sind linkslinear und konstruktorbasiert.

Die Aussage über die Entsprechung der Anzahl der [on]-Inferenzen mit der Anzahl der NN-Schritte folgt aus der Konstruktion. □

Man kann schließlich noch zeigen, daß die Reihenfolge der NN-Schritte der Lösung der entsprechenden [on]-Inferenzen entspricht. Dieser Sachverhalt wird nach dem folgenden Beispiel formuliert.

Beispiel 3.3.13. *(Zusammenhang zwischen NN- und OINC-Narrowingschritten)*
Es sei $\mathcal{R} = \{f(z) \to z, g(1) \to 1, h(1,1) \to 1\}$ ein TES und $h(f(g(x)), f(y))$ zu lösen.
Dann gibt es die NN-Sequenz

$$h(f(g(x)), f(y)) \rightsquigarrow h(g(x), f(y)) \rightsquigarrow_{\{x \mapsto 1\}} h(1, f(y)) \rightsquigarrow h(1, y) \rightsquigarrow_{\{y \mapsto 1\}} 1$$

und die entsprechende OINC'-Sequenz

$$\begin{array}{rll}
h(f(g(x)), f(y)) \approx s & \rightsquigarrow_{[in]} & [h(f(g(x)), f(y)), T_{1,1}, \varepsilon, 0] \approx [s, T_{1,2}, \varepsilon, 0] \\
 & \rightsquigarrow_{[on1]} & [f(g(x)), T_{1,1}, 1, 1] \approx [1, L_3, 1, 1], \\
 & & [f(y), T_{1,1}, 2, 1] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
 & \rightsquigarrow_{[on2]} & [g(x), T_{1,1}, 1 \cdot 1, 2] \approx [z, L_1, 1, 2], \\
 & & [z, R_1, 1, 2] \approx [1, L_3, 1, 1], \\
 & & [f(y), T_{1,1}, 2, 1] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
\Longrightarrow^2 & \rightsquigarrow_{[v], \{z \mapsto g(x)\}} & [g(x), R_1, 1, 2] \approx [1, L_3, 1, 1], \\
 & & [f(y), T_{1,1}, 2, 1] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
 & \rightsquigarrow_{[on3]} & [x, R_1, 1 \cdot 1, 3] \approx [1, L_2, 1, 3], \\
 & & [1, R_2, 1, 3] \approx [1, L_3, 1, 1], \\
 & & [f(y), T_{1,1}, 2, 1] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
\Longrightarrow^3 & \rightsquigarrow_{[v], \{x \mapsto 1\}} & [1, R_2, 1, 3] \approx [1, L_3, 1, 1], \\
 & & [f(y), T_{1,1}, 2, 1] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
 & \rightsquigarrow_{[d]} & [f(y), T_{1,1}, 2, 1] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
 & \rightsquigarrow_{[on4]} & [y, T_{1,1}, 2 \cdot 1, 2] \approx [z', L_1, 1, 2], \\
 & & [z', R_1, 2, 2] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
\Longrightarrow^4 & \rightsquigarrow_{[v], \{z' \mapsto y\}} & [y, R_1, 2, 2] \approx [1, L_3, 2, 1], \\
 & & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0] \\
\Longrightarrow^1 & \rightsquigarrow_{[v], \{y \mapsto 1\}} & [1, R_3, \varepsilon, 1] \approx [s, T_{1,2}, \varepsilon, 0]
\end{array}$$

Dabei ist mit $\Longrightarrow^2$ die [on]-2-gelöste Menge gekennzeichnet, die dem Narrowing mit Regel 1 an Position 1 *entspricht, mit $\Longrightarrow^3$ die [on]-3-gelöste Menge, die dem Narrowing mit Regel 2 an Position* 1 *entspricht, mit $\Longrightarrow^4$ die [on]-2-gelöste Menge, die dem Narrowing mit Regel 3 an Position* 2 *entspricht und mit $\Longrightarrow^1$*

die [on]-1-gelöste Menge, die dem Narrowing mit Regel 1 an der Wurzelposition entspricht.

■

Das zentrale Resultat dieses Abschnitts und ein zentrales Resultat dieser Arbeit ist

Satz 3.3.14.
Für jede NN-Sequenz $t \rightsquigarrow^+ t'$ gibt es eine OINC-Sequenz, in der die Regeln in derselben Reihenfolge an denselben Positionen angewendet, wie in der entsprechenden OINC-Sequenz die [on]-Schritte gelöst werden, d.h. bei Existenz einer NN-Ableitung

$$t_1 \rightsquigarrow_{p_1, L_1 \to R_1, \sigma_1} t_2 \rightsquigarrow \ldots \rightsquigarrow_{p_{n-2}, L_{n-2} \to R_{n-2}, \sigma_{n-2}} t_{n-1} \rightsquigarrow_{p_{n-1}, L_{n-1} \to R_{n-1}, \sigma_{n-1}} t_n$$

gibt es bei gegebener OINC-geeigneter Gleichung $t_1 \approx s$ mit $s \notin \mathcal{V}ar$ die OINC'-Inferenz

$$\begin{array}{lll} t_1 \approx s & \rightsquigarrow^* & \sigma_1\big([R_1, U_1, p_1, \tilde{n}_1] \approx X_1, E_1\big) \\ & \rightsquigarrow^* & \sigma_2 \circ \sigma_1\big([R_2, U_2, p_2, \tilde{n}_2] \approx X_2, E_2\big) \\ & \vdots & \\ & \rightsquigarrow^* & \sigma_{n-1} \circ \cdots \circ \sigma_1\big([R_{n-1}, U_{n-1}, p_{n-1}, \tilde{n}_{n-1}] \approx X_{n-1}, E_{n-1}\big), \end{array}$$

wobei für $1 \leq i < n$ U_i für einen Teilterm von s oder einen Teilterm einer linken Regelseite steht, X_i ein Quadrupel und E_i eine OINC'-Gleichungsmenge ist. Dabei sind die linkesten Gleichungen stets [on]-$\tilde{n}_i$-gelöst.
Daraus folgt insbesondere, daß die Anzahl der [on]-Inferenzen gleich der Anzahl der NN-Schritte ist.

Beweis
folgt direkt aus der Erweiterung des Satzes 3.3.11 auf den Kalkül OINC'. Die angegebenen inferierten Mengen mit einer Instanz einer rechten Regelseite als linke Seite der ersten Gleichung sind [on]-gelöst.

□

Bemerkung 3.3.15.

1. *Die Berechnung einer NN-Sequenz ausgehend von t_1 setzt voraus, daß $Head(t_1) \in \mathcal{F}$ ist. Das wurde in der Induktionsverankerung bei Anwendung des ersten [on]-Schritts an Position ε berücksichtigt (vgl. erneut Kap. 7 in [AEH94a]).*

2. *Wenn der letzte NN-Schritt an der Wurzelposition ε mit einer Regel $l \to r$ durchgeführt wird, dann besteht die schließlich von OINC inferierte Gleichungsmenge aus $r \approx s$ (die Anzahl der über der Narrowingposition liegenden Funktionssymbole ist Null).*

3. *Wenn $Head(t_1) \in \mathcal{F}$ ist und der letzte von NN berechnete Term t_n in Grundnormalform ist, dann gibt es eine OINC-Sequenz $t_1 \approx t_n \leadsto \square$ (der Beweis besteht aus der Fortsetzung der OINC-Berechnung, in der keine [on]-Inferenzen mehr nötig sind).*

3.3.2 Simulation von OINC durch NN

Die Simulation einer NN-Sequenz durch eine OINC-Sequenz wurde in den vorangegangenen Abschnitten gezeigt. Die Simulation von NN mit $\equiv$ durch OINC wird nun durch Reduktion der Simulation von OINC durch NN gezeigt. Dazu wird zunächst gezeigt (Satz 3.3.33), daß die von OINC berechneten Lösungen unvergleichbar bzgl. $\leq$ sind. Dann sind diese berechneten Lösungen immer "allgemeinste" Lösungen.

Die von NN berechneten Lösungen sind unvergleichbar in einem anderen Sinn ("unabhängig"): Für je zwei Lösungen σ_1 und σ_2 gibt es eine Variable x aus der initialen Gleichungsmenge, so daß $\sigma_1(x)$ und $\sigma_2(x)$ nicht unifizierbar sind. Es wird gezeigt, daß für derartige Lösungen bzgl. induktiv-sequentieller TES immer eine gemeinsame allgemeinere Substitution existiert, die dann berechnet wird.

Beispiel 3.3.16. *(Unvergleichbarkeit, Unabhängigkeit)*

Für ein TES $\{f(x,y) \to b, g(b) \to a\}$ und eine zu lösende Gleichung $g(f(x,y)) \equiv a$ sind $\{x \mapsto a\}$ und $\{y \mapsto b\}$ Lösungen, die unvergleichbar bzgl. $\leq$ sind, aber nicht unabhängig voneinander.

■

Aus dieser Unabhängigkeitseigenschaft folgt nun für Lösungen induktiv-sequentieller TES insbesondere die Unvergleichbarkeit zweier Substitutionen bzgl. $\leq$. Diese ist dann Schlüssel für den Beweis, daß die von OINC und NN berechneten Lösungen identisch sind (bzgl. strikter Gleichheit), woraus das gewünschte Simulationsresultat direkt gewonnen werden kann.

In Satz 3.3.11 wurde für jede NN-Sequenz, die eine Lösung berechnet, eine OINC-Sequenz konstruiert. Aus der Identität der berechneten Lösungen folgt dann die Simulation von NN durch OINC: *Nur* die in Satz 3.3.11 erzeugten OINC-Sequenzen berechnen eine Lösung, und damit ist die 1:1-Beziehung zwischen NN- und OINC-Sequenzen für erfolgreiche Ableitungen gezeigt. Die Simulation unendlicher oder fehlschlagender OINC- durch NN-Ableitungen wird nicht untersucht (in der Rückrichtung gilt das erzielte Simulationsresultat).

Die Argumentation stützt sich im wesentlichen darauf, daß man den von OINC errechneten Narrowingsequenzen bestimmte Reduktionen eindeutig zuordnen kann. Das sind die sog. *Standardreduktionen* ([HL91], vgl. [IN94], Def. 4.13). Vereinfacht gesagt, werden bei orthogonalen TES in Standardableitungen nur diejenigen Redexe kontrahiert, die auch tatsächlich kontrahiert werden müssen. Standardreduktionen haben die Eigenschaft, daß sie eindeutig sind:

Satz 3.3.17. *(Theorem 3.19 in [HL91])*
Jede Reduktionsklasse bzgl. OTES enthält genau eine Standardreduktion.

Im folgenden wird nun die Unvergleichbarkeit der von OINC berechneten Substitutionen bezüglich $\leq$ gezeigt. Dazu wird die Tatsache benötigt, daß OINC-Ableitungen dieselben Regeln an denselben Positionen in derselben Reihenfolge anwenden wie die entsprechende Standardreduktion.
Dazu werden erneut die zwei folgenden Notationen benötigt: $\top$ bezeichnet (generisch) eine Folge von einem oder mehreren *true*s, und es ist $\mathcal{R}_+ = \mathcal{R} \cup \{x \approx x \to \mathit{true}\}$ ($\mathcal{R}_+$ ist nicht orthogonal). Zunächst muß die Definition der Standardreduktion, die sich auf Terme bezieht, auf Gleichungsmengen erweitert werden:

Definition 3.3.18. *(Standardreduktion für Gleichungen, Def. 4.13 und 4.14 in [IN94])*

- e_0 *sei eine Gleichung. Für* $k \geq 0$ *ist die Reduktion*

$$e_0 \to_{\mathcal{R}} e_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} e_k \to_{\mathcal{R}_+} \mathit{true}$$

  eine Standardreduktion, falls $e_0 \to_{U_1} e_1 \to_{U_2} \cdots \to_{U_k} e_k$ *eine Standardreduktion ist, wobei für* $1 \leq i \leq k$ $U_i = \{u_i\}$ *und* $e|_{u_i}$ *das Redex ist, das in der Reduktion* $e_{i-1} \to_{\mathcal{R}} e_i$ *kontrahiert wird.*

- *Eine Standardreduktion* $A : G \to^*_{\mathcal{R}_+} \top$ *ist für eine initiale Gleichungsmenge* G *induktiv definiert:*

  – $A : \square \to \square$ *ist eine Standardreduktion*
  – $A : e, E \to^*_{\mathcal{R}_+} \mathit{true}, E \to^*_{\mathcal{R}_+} \top$ *ist eine Standardreduktion, falls*
    1. $e \to^*_{\mathcal{R}_+} \mathit{true}$ *eine Standardreduktion ist und*
    2. $E \to^*_{\mathcal{R}_+} \top$ *eine Standardreduktion ist.*

$\triangle$

Eine Standard-Narrowing-Ableitung (kurz SNC-Ableitung) wird nun entsprechend definiert:

Definition 3.3.19. *(Standard-Narrowing-Ableitung, Def. 4.15 in [IN94])*
$\mathcal{R}$ sei ein OTES.

- *e_0 sei eine rechtsnormale initiale Gleichung (oder eine von einem OI-Narrowing-Kalkül inferierte, wobei die initiale Gleichung rechtsnormal ist).*

  Eine Narrowing-Ableitung $e_0 \leadsto_{\sigma_1} e_1 \leadsto_{\sigma_2} \cdots \leadsto_{\sigma_k} e_k \leadsto_{\sigma_{k+1}} true$,

  wobei nur im letzten Schritt die Regel $x \approx x \to true$ angewendet wurde, heißt mit $k \geq 0$ und $e_k = s_k \approx t_k$ "Standard-Narrowing-Ableitung", falls

  1. *die entsprechende Reduktion*

     $$\mu_0(e_0) \to_{\{1 \cdot u_1\}} \mu_1(e_1) \to \cdots \to_{\{1 \cdot u_k\}} \mu_k(e_k) \to true$$

     mit $\mu_i = \sigma_{k+1} \circ \cdots \circ \sigma_{i+1}$ für $i = 0, \ldots, k$ eine Standardreduktion ist und

  2. *$min(\{u_1, \ldots u_k\}) \subseteq \overline{\mathcal{O}}(t_k)$ ist, wobei $min(U) = \{v \in U \big| \nexists u \in U.u \prec v\}$ gilt und $\{u_1, \ldots u_k\}$ eine Multimenge mit den Positionen $u_1, \ldots, u_k$ ist.*

- *G sei eine rechtsnormale initiale Gleichungsmenge (oder eine von einem OI-Narrowing-Kalkül inferierte, wobei die initiale Gleichungsmenge rechtsnormal ist). Eine Standard-Narrowing-Ableitung für eine erfolgreiche Narrowing-Ableitung $A : G \leadsto^*_\eta \top$ ist induktiv definiert:*

  1. *Die leere Ableitung ist eine Standard-Narrowing-Ableitung.*
  2. *$A : \top, e, E \leadsto^*_\sigma \top, \sigma(E) \leadsto^*_\vartheta \top$ (die linkeste Gleichung wird immer zuerst gelöst) ist eine Standardableitung, falls*
     - *$e \leadsto^*_\sigma \top$ (aus A) eine Standard–Narrowing–Ableitung ist und*
     - *$\sigma(E) \leadsto^*_\vartheta \top$ (aus A) eine Standard-Narrowing-Ableitung ist.*

$\triangle$

Bemerkung 3.3.20.

1. *Hier wird nicht von OINC-geeigneten Gleichungsmengen gesprochen, weil die Definition unabhängig vom verwendeten Kalkül sein soll. Ein "OI-Kalkül" ist ein Narrowingverfahren, das outside-in-Ableitungen berechnet (s.u.).*

2. *$\rightsquigarrow$ bezeichnet keine OINC-Inferenz, sondern einen Narrowingschritt (s. Kap 2).*

3. *Das Präfix* 1 *der Positionen in der Reduktion der ersten Bedingung wird hinzugefügt, weil $Head(e_i) =\approx$ ist und die rechte Seite der Gleichung in Normalform ist.*

4. *Die Zuordnung von Narrowing-Ableitungen zu Standardreduktionen ist injektiv, wegen Bedingung (2) des ersten Teils der Definition aber nicht surjektiv (s. Beispiel 4.2 in [IN94]; vgl. den Bezug mit NN in [AEH94a], z.B. Beispiel 4).*

In [IN94] wird der Kalkül OINC anhand einer speziellen Klasse von Narrowing-Ableitungen definiert, nämlich den *leftmost-outside-in-Ableitungen* (Def. 4.9 in [IN94]). Das Charakteristikum der LOI-Ableitungen liegt u.a. darin, daß die Positionen, an denen Narrowing-Schritte durchgeführt werden, in der Reihenfolge leftmost-outside-in sortiert sind. Das ist allerdings nicht die Definition von LOI; diese erfolgt in Def. 4.5 und 4.9 in [IN94] über eine (Vereinfachungs-)Ordnung $\overset{\mathrm{OI}}{\rhd}$ mit "OI" für "outside-in" auf Narrowingableitungen (die die Standardeigenschaft von Reduktionen nicht erhält; vgl. Kap. 4.4, insb. Bsp. 4.2 in [IN94]). Für die hier verfolgten Ziele ist es ausreichend zu wissen, daß LOI-Ableitungen die "größten" bzgl. einer Ordnung ($\overset{\mathrm{OI}}{\rhd}$) sind. Auf die umfangreiche Definition wird hier verzichtet, weil sie ohne Modifikation aus [IN94] übernommen würde.

Um zu zeigen, daß einer LOI-Narrowing-Ableitung eine Standardreduktion mit Anwendung derselben Regeln an denselben Positionen entspricht, werden die folgenden Lemmata und Sätze (Beweis in [IN94]) benötigt:

Satz 3.3.21. *(Theorem 4.2 in [IN94])*
Für eine erfolgreiche Narrowing-Ableitung A gilt: A ist eine SNC-Ableitung gdw. A ist LOI.

Lemma 3.3.22. *(Lifting-Lemma 4.9 für Narrowing-Ableitungen in [IN94])*
Wenn es für ein TES $\mathcal{R}$, eine initiale Gleichungsmenge G, eine normalisierte Substitution ϑ und eine Variablenmenge $\mathcal{W} \supseteq \mathcal{V}ar(G) \cup dom(\vartheta)$ eine Reduktion

$$\vartheta(G) \to^*_{\mathcal{R}_+} G'$$

gibt, dann gibt es eine Gleichungsmenge G'' und Substitutionen ϑ' und σ, so daß

- $G \rightsquigarrow^*_\sigma G'$,
- $\vartheta'(G'') = G'$,

- *$\vartheta \circ \sigma = \vartheta[\mathcal{W}]$ und*
- *ϑ' normalisiert ist.*

In beiden Ableitungen werden dieselben Regeln an denselben Positionen der korrespondierenden Gleichung in jedem Schritt angewendet.

Lemma 3.3.23. *(Proposition 4.3 in [IN94])*
Es sei $\mathcal{R}$ ein OTES und G eine rechtsnormale initiale Gleichungsmenge. Wenn es eine erfolgreiche Narrowing-Ableitung $G \rightsquigarrow^*_\vartheta \top$ gibt, dann kann im Bildbereich der Lösung $\vartheta \restriction_{\mathcal{V}ar(G)}$ kein Narrowing-Schritt durchgeführt werden, d.h. für alle $x \in \operatorname{dom}(\vartheta) \cap \mathcal{V}ar(G)$ gilt, das in $\vartheta(x)$ kein Narrowing-Schritt durchgeführt werden kann.

Lemma 3.3.24. *(Lemma 4.10 in [IN94])*
Es sei $\mathcal{R}$ ein OTES, ϑ eine normalisierte Substitution und G eine initiale Gleichungsmenge. $A : \vartheta(G) \rightarrow^*_{\mathcal{R}_+} \top$ und $A' : \sigma(G) \rightarrow^*_{\mathcal{R}_+} \top$ seien Reduktionen mit $\xi \circ \sigma = \vartheta$, wobei ξ normalisiert ist und in beiden Reduktionen dieselbe Regel an derselben Position in derselben Gleichung angewendet wird.
Wenn A eine Standardreduktion ist, dann ist A' eine Standardreduktion.

Satz 3.3.25. *(Standardisierungstheorem für Narrowingableitungen, Theorem 4.3 in [IN94])*
Es sei $\mathcal{R}$ ein OTES und G eine rechtsnormale initiale Gleichungsmenge. Wenn es eine erfolgreiche Narrowing-Ableitung $G \rightsquigarrow^*_\vartheta \top$ gibt, dann gibt es eine LOI-Narrowing-Ableitung $G \rightsquigarrow^*_\sigma \top$ mit $\sigma \leq \vartheta[\mathcal{V}ar(G)]$.

Beweis ([IN94])
Lemma 3.3.23 liefert die Tatsache, daß im Bildbereich von $\eta = \vartheta[\mathcal{V}ar(G)]$ kein Narrowingschritt durchgeführt werden kann. Wegen der Korrespondenz zwischen Narrowing-Ableitungen und Reduktionen gibt es eine Reduktion $\eta(G) \rightarrow^*_{\mathcal{R}_+} \top$. Dann gibt es auch eine Standardreduktion $A : \eta(G) \rightarrow^*_{\mathcal{R}_+} \top$. Das Lifting-Lemma 3.3.22 zeigt die Existenz einer erfolgeichen Narrowing-Ableitung $B : G \rightsquigarrow^*_\sigma \top$ mit $\xi \circ \sigma = \eta[\mathcal{V}ar(G)]$ für eine normalisierte Substitution ξ. B' sei die Reduktion $\sigma(G) \rightarrow^*_{\mathcal{R}_+} \top$, die B entspricht. Die Anwendung von Lemma 3.3.24 auf die Reduktionen A und B' gestattet die Schlußfolgerung, daß B' eine Standardreduktion ist, und damit ist B eine SNC-Ableitung nach Definition. Satz 3.3.21 weist nach, daß B LOI ist.

□

Ein Schlüsselargument für den Beweis der Unabhängigkeit der von OINC errechneten Lösungen wird durch das folgende Lemma geliefert:

Lemma 3.3.26.
Wenn es für eine initiale Gleichungsmenge G eine LOI-Narrowing-Ableitung $G \rightsquigarrow_\sigma^* \top$ gibt, dann ist die entsprechende Reduktion $G \rightarrow_{\mathcal{R}_+}^* \top$ bei Anwendung derselben Regeln an denselben Positionen der jeweils korrespondierenden Gleichung in derselben Reihenfolge eine Standardreduktion.

Beweis
folgt direkt aus der Konstruktion im Beweis des Satzes 3.3.25: Dort ist B LOI, die entsprechende Reduktion B' ist eine Standardreduktion, und in B' werden nach Konstruktion dieselben Regeln in denselben Situationen wie in B angewendet. Man beachte, daß das Narrowing bzgl. $\mathcal{R}$ und die Reduktion bzgl. $\mathcal{R}_+$ stattfindet. Außerdem sei erneut darauf hingewiesen, daß die erfolgten Definitionen zunächst unabhängig von OINC sind und $\rightsquigarrow$ insbesondere keine OINC-Inferenzen, sondern Narrowingschritte (Kap. 2) bezeichnet.

$\square$

Das Interesse an LOI-Ableitungen resultiert daraus, daß OINC genau diese erzeugt:

Lemma 3.3.27.
Für eine initiale Gleichungsmenge G sind die von OINC erzeugten Ableitungen $G \rightsquigarrow^* \square$ LOI.

Beweis
folgt aus der Definition von OINC und LOI (bzw. $\overset{\mathrm{OI}}{\rhd}$). Vgl. Kap. 5 in [IN94].

$\square$

Man mache sich klar, daß OINC die Narrowingschritte $\rightsquigarrow$ durch "feinere" Inferenzregeln ersetzt (Kap.5 in [IN94]), die mit den einzelnen Komponenten der Ordnung $\overset{\mathrm{OI}}{\rhd}$ übereinstimmen.
Es ist ausreichend, sich auf Konstruktorsubstitutionen zu beschränken, denn wie NN berechnet auch OINC ausschließlich solche:

Lemma 3.3.28.
Es sei G eine Gleichungsmenge, in der alle rechten Regelseiten Grundkonstruktorterme sind.

1. *In konstruktorbasierten orthogonalen TES (KBO-TES) gilt für alle OINC-Sequenzen $G \rightsquigarrow^* E$:*
   - *für alle $l \approx r \in E$ ist $\mathcal{V}ar(G) \cap \mathcal{V}ar(r) = \emptyset$,*
   - *die Variablenmengen aller rechten Regelseiten sind paarweise disjunkt und*

- *für alle $l \approx r \in E$ ist r ein Konstruktorterm oder eine Variable.*

2. *In KBO-TES gilt für alle von OINC errechneten Substitutionen σ : $\sigma \restriction_{\mathcal{V}ar(G)}$ ist eine Konstruktorsubstitution.*

Beweis
von (1) durch Induktion über die Länge der OINC-Ableitung.
Die Induktionsverankerung (leere Sequenz) ist trivial.
Induktionsschritt:
Es sind die einzelnen Inferenzregeln von OINC zu untersuchen. Die Induktionsvoraussetzung gelte für $E = s \approx t, E'$.
1. Fall: [on]
Dann ist $Head(s) \in \mathcal{F}$ (Stelligkeit n) und $t \notin \mathcal{V}ar$, und es ist die Inferenz

$$s \approx t, E' \rightsquigarrow_{[on]} s|_1 \approx l|_1,\ \dots, s|_n \approx l|_n, r \approx s, E'$$

für eine frische Regel $l \to r$ möglich. Die Regel ist frisch; demnach werden auf den rechten Gleichungsseiten keine Variablen aus $\mathcal{V}ar(G)$ eingeführt, und die neuen Variablen tauchen nicht in E' auf. KBO-TES sind wegen der Orthogonalität links-linear; also sind die Variablenmengen der neuen rechten Gleichungsseiten disjunkt. Das TES ist nach Voraussetzung konstruktorbasiert; dann sind die l_i für $i \leq n$ Konstruktorterme. Da keine Variablen gebunden werden, wird E' nicht modifiziert. Also gilt die Behauptung für die gesamte inferierte Gleichungsmenge.
2.Fall: [d]
Dann ist $s = f(s_1,\ \dots, s_n)$ und $t = f(t_1,\ \dots, t_n)$ für $n \geq 0$ und $f \in \mathcal{C} \cup \mathcal{F}$, und es ist die Inferenz

$$s \approx t, E' \rightsquigarrow_{[d]} s_1 \approx t_1,\ \dots, s_n \approx t_n, E'$$

möglich. Da die neu eingeführten rechten Gleichungsseiten Unterterme der alten rechten Gleichungsseite t sind und keine Variablen gebunden werden, gilt die Behauptung für die inferierte Gleichungsmenge trivialerweise.
3.Fall: [v1]
Dann ist $t \in \mathcal{V}ar$. Da die Variable t nach Voraussetzung in keiner rechten Regelseite aus E' vorkommt, gilt die Behauptung auch für die inferierte Gleichungsmenge $\{t \mapsto s\}(E')$.
4.Fall: [v2]
Dann ist $s \in \mathcal{V}ar$ und $t \notin \mathcal{V}ar$, und die Induktionsvoraussetzung gilt insbesondere für t, d.h. t ist ein Konstruktorterm, dessen Variablenmenge disjunkt zu den Variablenmengen aller rechten Gleichungsseiten und zu $\mathcal{V}ar(G)$ ist. Dann gilt die Behauptung auch für die inferierte Gleichungsmenge $\{s \mapsto t\}(E')$, weil alle Vorkommen von s auf rechten Gleichungsseiten durch t ersetzt werden, d.h. durch einen Konstruktorterm, dessen Variablenmenge disjunkt zu den Variablenmengen aller rechten Gleichungsseiten und zu $\mathcal{V}ar(G)$ ist.

zu (2): Aus dem ersten Teil des Lemmas ergibt sich, daß Variablen aus G nur auf linken Gleichungsseiten auftreten und damit auch nur mittels einer [v2]-Inferenz an rechte Gleichungsseiten gebunden werden können. Alle rechten Gleichungsseiten sind Konstruktorterme, also werden Variablen auf linken Gleichungsseiten (sofern überhaupt) an Konstruktorterme gebunden, und damit können insbesondere die Variablen aus G nur an Konstruktorterme gebunden werden.

□

Für nicht konstruktorbasierte Systeme gilt, daß der Bildbereich der errechneten Substitutionen in Normalform ist.
Es wird schließlich noch der Zusammenhang zwischen Standardreduktionen mit vergleichbaren Lösungen benötigt:

Lemma 3.3.29.
Es sei G eine rechtsnormale initiale Gleichunsgmenge und $\mathcal{R}$ ein OTES.
Wenn es zwei Standardreduktionen $A : G \to^*_{\mathcal{R}_+} \top$ und $B : \vartheta(G) \to^*_{\mathcal{R}_+} \top$ für eine Konstruktorsubstitution ϑ gibt, dann werden in beiden Ableitungen dieselben Regeln an denselben Positionen in derselben Reihenfolge angewendet.

Beweis
In einem Reduktionsschritt $t_1 \to_{p,l\to r,\eta} t_2$ aus A ist p das leftmost-outermost Redex, weil A eine Standardableitung ist. Dann ist p auch das leftmost-outermost Redex in $\sigma(t_1)$ in der Reduktion $\vartheta(t_1) \to_{p',l'\to r',\eta'} \vartheta(t_2')$ für jede Konstruktorsubstitution ϑ, weil durch Anwendung der Substitution kein neues leftmost-outermost Redex entsteht.
Nach Definition von $\to_{\mathcal{R}}$ gilt [BA95]

$$t \to_{\mathcal{R}} s \Leftrightarrow \exists p \in \overline{\mathcal{O}}(t) \exists \sigma \in Subst \exists l \to r \in \mathcal{R}. t|_p = \sigma(l) \wedge s = t[\sigma(r)]_p$$

bei Verwendung einer Regel $l \to r$.
In obigem Ersetzungsschritt ist demnach $\eta(l) = t_1|_p$ und $t_2 = t_1[\eta(r)]_p$. Dann gibt es auch η' mit $\eta'(l) = \vartheta(t_1)|_p$, nämlich $\eta' = \eta \circ \vartheta$. Damit kann dieselbe Regel in beiden Fällen angewendet werden, d.h. $l \to r = l' \to r'$, was insbesondere $t_2' = t_2$ impliziert.
Der Beweis erfolgt dann durch vollständige Induktion über die Länge der Ableitung. Man mache sich klar, daß die Regel $x \approx x \to true$ nur im jeweils letzten Schritt der Ableitungen der einzelnen Gleichungen aus G Anwendung findet.

□

Damit kann nun die Unvergleichbarkeit der Lösungen nachgewiesen werden:

Lemma 3.3.30.
Die von OINC errechneten Lösungen für eine initiale rechtsnormale Gleichungsmenge G sind unvergleichbar bzgl. $\leq$, wenn man die Substitutionen auf die Variablen aus G einschränkt.

Beweis
OINC erzeugt LOI-Sequenzen nach Lemma 3.3.27. Nach Satz 3.3.21 gilt: A ist LOI gdw. A ist eine SNC-Ableitung. Eine Narrowingableitung ist nach Definition 3.3.19 eine SNC-Ableitung, falls die ihr (injektiv!) zugeordnete Reduktion eine Standardreduktion ist. Standardreduktionen sind eindeutig nach Satz 3.3.17. Es seien nun ϑ und σ zwei von OINC errechnete Lösungen mit $\sigma \leq \vartheta[\mathcal{V}ar(G)]$. Eine Narrowing-Ableitung A berechne ϑ und eine Narrowing-Ableitung B berechne σ mit $A \neq B$. Die zugeordneten Reduktionen A' und B' sind gemäß Lemma 3.3.26 Standardreduktionen, weil A und B LOI sind. Lemma 3.3.29 liefert $A' = B'$ wegen $\sigma \leq \vartheta[\mathcal{V}ar(G)]$. Da die Zuordnung von Narrowing-Ableitungen zu Reduktionen injektiv ist, folgt $A = B$. Das ist ein Widerspruch zur Annahme $A \neq B$.

□

Außerdem gilt

Lemma 3.3.31.
Es seien σ_1 und σ_2 zwei (Konstruktor-)Lösungen einer Anfrage G bzgl. eines induktiv-sequentiellen TES, die unvergleichbar bzgl. $\leq$ und abhängig voneinander sind. Dann gibt es eine Substitution ϑ mit $\vartheta \leq \sigma_1$ und $\vartheta \leq \sigma_2$, so daß ϑ ebenfalls Lösung für G ist.

Beweis
Es seien σ_1 und σ_2 zwei Konstruktorsubstitutionen, die Lösungen einer initialen Gleichungsmenge G und abhängig voneinander sind, d.h. für alle $x \in \mathcal{V}ar(G)$ gelte, daß $\sigma_1(x)$ und $\sigma_2(x)$ unifizierbar sind. Außerdem gelte weder $\sigma_1 \leq \sigma_2$ noch $\sigma_2 \leq \sigma_1$.
Dann berechne NN Lösungen ϑ_1 und ϑ_2 mit $\vartheta_1 \leq \sigma_1[\mathcal{V}ar(G)]$ und $\vartheta_2 \leq \sigma_2[\mathcal{V}ar(G)]$ (evtl. wird nur eine Lösung berechnet).
1. Fall: Wenn $\vartheta_1 = \vartheta_2$ ist, dann setzt man $\vartheta = \vartheta_1 = \vartheta_2$, und die Behauptung ist nachgewiesen.
2. Fall: Dann sind ϑ_1 und ϑ_2 unabhängig voneinander, weil sie von NN berechnet wurden; d.h. es gibt ein $\overline{x} \in \mathcal{V}ar(G)$, so daß $\vartheta_1(\overline{x})$ und $\vartheta_2(\overline{x})$ nicht unifizierbar sind. Es gelte $\sigma_1 = \varphi_1 \circ \vartheta_1$ und $\sigma_2 = \varphi_2 \circ \vartheta_2$ für geeignete Substitutionen φ_1 und φ_2.
Aus der Abhängigkeit von σ_1 und σ_2 folgt nun aber, daß es insbesondere eine Substitution η mit $\eta(\sigma_1(\overline{x})) = \eta(\sigma_2(\overline{x}))$ gibt. Einsetzen liefert dann

$$\eta(\varphi_1(\vartheta_1(\overline{x}))) = \eta(\varphi_2(\vartheta_2(\overline{x}))).$$

Da $\vartheta_1(\overline{x})$ und $\vartheta_2(\overline{x})$ nach Voraussetzung nicht unifizierbar sind, gibt es ein minimales p mit $p \in \overline{\mathcal{O}}(\vartheta_1(\overline{x})) \cap \overline{\mathcal{O}}(\vartheta_2(\overline{x}))$, so daß $Head(\vartheta_1(\overline{x})|_p) \neq Head(\vartheta_2(\overline{x})|_p)$ und $Head(\vartheta_1(\overline{x})|_q) = Head(\vartheta_2(\overline{x})|_q)$ für alle $\varepsilon \preceq q \prec p$ gilt.
Da nach Konstruktion die Symbole an p und über p nicht variabel sind, gilt für alle Substitutionen ζ und ξ, daß $Head(\xi(\vartheta_1(\overline{x}))|_p) \neq Head(\zeta(\vartheta_2(\overline{x}))|_p)$ und damit auch $\xi(\vartheta_1(\overline{x})) \neq \zeta(\vartheta_2(\overline{x}))$ ist.

Setzt man nun $\xi = \eta \circ \varphi_1$ und $\zeta = \eta \circ \varphi_2$, so folgt ein Widerspruch zur Annahme, daß σ_1 und σ_2 abhängig voneinander sind.
Also kann dieser Fall ($\vartheta_1 \neq \vartheta_2$) nicht auftreten.

□

Lemma 3.3.32.
Die von OINC berechneten Lösungen sind allgemeinste Lösungen, d.h. bei gegebener errechneter Lösung σ

1. *gibt es kein nicht von OINC berechnetes $\vartheta < \sigma$, das ebenfalls Lösung ist und*

2. *in der Menge der von OINC berechneten Lösungen gibt es kein ϑ mit $\vartheta > \sigma$.*

Beweis

1. Folgt aus der Vollständigkeit von OINC.

2. Gäbe es ein solches ϑ, so wären σ und ϑ vergleichbar bzgl. $\leq$, im Widerspruch zu Lemma 3.3.30.

□

Lemma 3.3.33.
Die von OINC berechneten Lösungen sind paarweise unabhängig.

Beweis
σ_1 und σ_2 seien zwei verschiedene, von OINC berechnete Substitutionen, die voneinander abhängig sind. Nach Lemma 3.3.31 gibt es eine Lösung ϑ mit $\vartheta \leq \sigma_1$ und $\vartheta \leq \sigma_2$. OINC berechnet allgemeinste Lösungen nach Lemma 3.3.32. Damit würde OINC ein $\vartheta' \leq \vartheta$ und nicht σ_1 oder σ_2 berechnen.

□

Lemma 3.3.34.
Die von OINC und NN berechneten Lösungen sind identisch bis auf Variablenumbenennung.

Beweis
$\Sigma = \{\sigma_1, \sigma_2, \dots\}$ seien alle Lösungen einer Gleichungsmenge G (Konstruktorsubstitutionen). $\Theta = \{\vartheta_1, \vartheta_2, \dots\}$ seien die von NN berechneten Lösungen. Dann sind die ϑ_i paarweise unabhängig und unvergleichbar bzgl. $\leq$; deshalb geht durch die Notation als Menge keine Information "verloren". Die Vollständigkeit

von NN liefert die Tatsache, daß jedes σ_i von einem ϑ_j subsumiert wird. Θ ist also eine Art Basis für Σ. Weil Θ von NN berechnet wurde, ist Θ eindeutig bis auf Variablenumbenennung.
Ω sei nun die Menge der von OINC errechneten Lösungen (auch hier bereitet die Mengennotation keine Schwierigkeiten, weil nachgewiesen wurde, daß OINC Lösungen für induktiv-sequentielle Systeme nicht mehrfach berechnet). $\Theta \subseteq \Omega$ folgt aus dem Simulationssatz 3.3.11. Es sei nun $\zeta \in \Omega$ und $\zeta \notin \Theta$, d.h. eine von OINC, nicht aber von NN berechnete Lösung. Da nach Lemma 3.3.33 ζ unabhängig von allen Elementen aus Ω ist und zu keinem Element in Subsumtionsbeziehung steht, ist ζ wegen $\Theta \subseteq \Omega$ auch unabhängig von allen Elementen aus Θ. Dann berechnet NN ein $\xi \leq \zeta$, das Lösung von G ist, also $\xi \in \Theta$. $\xi \in \Theta$ impliziert nun $\xi < \zeta$ wegen $\zeta \notin \Theta$. Damit ist ξ echt allgemeiner als ζ, und dann würde OINC ξ und nicht ζ berechnen, ein Widerspruch zur Annahme.
Also ist $\Theta = \Omega$.

□

Damit ist nun auch die Simulation von OINC durch NN gezeigt worden. Es folgt

Satz 3.3.35.
In einem TES mit Regeln für die strikte Gleichheit kann jede NN-Sequenz $t \equiv s \rightsquigarrow^* true$ *durch genau eine OINC-Sequenz* $t \equiv s \approx true \rightsquigarrow^* \Box$ *simuliert werden, und die Rückrichtung gilt auch.*

Beweis
Die von beiden Kalkülen errechneten Lösungen sind identisch (modulo Variablenumbenennung) nach Lemma 3.3.34. Die errechneten Substitutionen sind in beiden Fällen Konstruktorsubstitutionen (Lemma 3.3.28).
Für jede NN-Sequenz wurde eine OINC-Sequenz erzeugt. Da OINC keine zusätzlichen Lösungen berechnet, sind die erzeugten OINC-Sequenzen wegen der Vollständigkeit von NN die einzigen, die eine Lösung liefern. Damit ist jeder NN-Ableitung genau eine OINC-Ableitung zugeordnet worden.
Die OINC-Sequenzen haben, sofern sie erfolgreich sind, also immer die im Simulationsbeweis angegebene Form.

□

3.4 Vergleich der Komplexitäten

In den letzten Abschnitten wurde dargelegt, daß zwischen OINC und NN eine wechselseitige Simulationsbeziehung (erfolgreiche Ableitungen) besteht. In bezug auf den praktischen Einsatz eines der beiden Verfahren ist die Frage von Interesse, welches von beiden "komplexer" ist, d.h. mehr Hauptspeicher oder mehr Zeit benötigt. Dazu werden zunächst grundsätzliche Überlegungen angestellt und dann experimentelle Resultate angegeben.

3.4.1 Grundsätzliche Überlegungen

Da die einzelnen OINC-Inferenzen auf einer technisch viel feineren Ebene liegen als die NN-Schritte, stellt sich die Frage nach einem geeigneten Komplexitätsmaß. Die Länge der Ableitung ist offensichtlich nicht geeignet.

Der fundamentale Unterschied zwischen beiden Kalkülen ist der Zeitpunkt der Auswahl der Regeln (in induktiv-sequentiellen TES kann wegen der Linearität und Konstruktorbasiertheit der linken Regelseiten kein Nichtdeterminismus zwischen der [on]- und der [d]-Inferenz bestehen, wenn man in initialen Gleichungen auf rechten Seiten keine Funktionssymbole zuläßt).
OINC muß grundsätzlich *alle* Regeln ausprobieren, sobald ein [on]-Schritt durchgeführt werden kann. In NN wird diese Entscheidung so lange wie möglich verzögert.
Damit ist OINC insbesondere nichtdeterministisch auf Grundtermen, im Unterschied zu NN [AEH94a]. In NN besteht der einzige Nichtdeterminismus in der Auswahl des Unterbaums im Fall $R_3^{\lambda'}$, wenn mehrere Muster mit dem auszuwertenden (Unter-)Term unifiziert werden können.

In [Han95] wird (insbesondere) eine effiziente Implementierung von NN angegeben, die auf einer Übersetzung der Regeln in PROLOG-Programme beruht. Dazu wird für jede Verzweigungsposition eines jeden definierenden Baums ein eigenständiges Prädikat angelegt.
Die folgenen (NN betreffenden) Komplexitätsbetrachtungen werden sich an dieser Implementierung orientieren, sind aber verallgemeinerbar. Außerdem wird später davon ausgegangen, daß der Nichtdeterminismus beider Kalküle über (z.B. PROLOG's) Backtrackingmechanismen realisert wird .

Beispiel 3.4.1. *(Übersetzung von NN, [Han95])*
Dem TES $\mathcal{R} = \{0 + x \rightarrow x, s(x) + y \rightarrow s(x + y), 0 \leq x \rightarrow true, s(x) \leq 0 \rightarrow false, s(x) \leq s(y) \rightarrow x \leq y\}$ *entspricht das (automatisch generierte, nicht optimierte) PROLOG-Programm*

```
A===B :- hnf(A,HA), hnf(B,HB), seq(HA,HB).
seq(0,0).
seq(s(A),s(B)) :- A===B.
seq(false,false).
seq(true,true).

hnf(T,T) :- var(T), !.
hnf(A+B,H) :- !, +(A,B,H).
hnf(leq(A,B),H) :- !, leq(A,B,H).
hnf(T,T).

+(A,B,R) :- hnf(A,HA), '+_1'(HA,B,R).
```

```
'+_1'(0,B,R) :- hnf(B,R).
'+_1'(s(A),B,R) :- hnf(s(A+B),R).

leq(A,B,R) :- hnf(A,HA), leq_1(HA,B,R).
leq_1(0,_,R) :- hnf(true,R).
leq_1(s(A),B,R) :- hnf(B,HB), leq_1_2(s(A), HB, R).
leq_1_2(s(_),0,R) :- hnf(false,R).
leq_1_2(s(A),s(B),R) :- hnf(leq(A,B),R).
```

Dabei sind `===` *und* `seq` *Klauseln für die strikte Gleichheit, und* `hnf` *berechnet die Kopfnormalform eines Terms.*

■

Die Definition der Narrowingfunktion λ ist in bezug auf die Implementierung in zwei Hinsichten irreführend.
Zum einen legt sie nahe, daß bei Verzweigung in einen Unterbaum der auszuwertende Term tatsächlich mit den Mustern der Wurzelknoten aller Unterbäume unifiziert wird.
Zum anderen wird gemäß dieser Definition für jeden Schritt wieder an der Wurzelposition des auszuwertenden Terms begonnen, was ebenso unnötig ist (vgl. Lemma 3.3.7 über identische Präfixe der λ-Aufruffolgen zweier aufeinander folgender Schritte).

Die zitierte Implementierung umgeht beide Probleme:

1. Die Unifikation mit den Mustern der Wurzeln der Unterbäume kann über Indizierung des Teilterms an der Verzweigungsposition effizient durchgeführt werden. Insbesondere muß die Unifikation nur mit den Formalparametern *eines* Prädikats vorgenommen werden, wenn der entsprechende Aktualparameter keine Variable ist (d.h. der Teilterm an der der Verzweigungsposition entsprechenden Stelle ist keine Variable).

2. PROLOG's Backtrackingmechanismus löst das zweite Problem direkt. Wenn Narrowing an einer Position durchgeführt wurde, wird nicht wieder von vorne an der Wurzelposition des neuen (resultierenden) Terms begonnen, sondern

   - im Term weiter nach unten verzweigt, falls die nächste Narrowingposition weiter unten liegen sollte oder
   - schrittweise im Term nach oben zurückgesprungen, falls die nächste Narrowingposition über oder rechts der letzten Position liegt.

Das Absteigen in einem DT erfordert nun die Unifikation der Formalparameter der Prädikate für die verschiedenen Verzweigungspositionen mit den Argumenten der auszuwertenden Funktion. Demzufolge müssen für einen vollständigen

Abstieg in einem DT der Tiefe t insgesamt t Unifikationen mit Mustern durchgeführt werden, und die Entscheidung für das "richtige" Muster – von denen es mehrere geben kann – erfolgt in konstanter Zeit.
Wenn man nun nur "unterschiedliche" Verzweigungen betrachtet, d.h. gleiche Anfangsstücke von λ-Aufruffolgen ignoriert (nach Lemma 3.3.7 sind diese bis zum tiefsten gemeinsamen Funktionssymbol zweier aufeinanderfolgender Narrowingschritte identisch), so ist – bei gewünschten resultierenden Konstruktortermen – die Anzahl dieser unterschiedlichen Verzweigungen gleich der Anzahl der Narrowingschritte. Das liegt daran, daß NN in orthogonalen induktiv-sequentiellen, also insbesondere konstruktorbasierten TES rechnet. Wenn dann an einer Position in einen anderen Baum verzweigt wird, so muß der Unterterm an dieser Stelle im Verlauf der Rechnung zu einem Konstruktorterm abgeleitet werden, damit auch der Gesamtterm zu einem Konstruktorterm abgeleitet werden kann. Es ist hier ausreichend, resultierende Konstruktorterme zu betrachten, weil zum einen strikte Gleichungen zugrundeliegen und zum anderen sonst keine direkte Entsprechung mit OINC besteht (denn die OINC-Gleichheit $\approx$ gibt es in NN nicht).

Demzufolge kann der Aufwand für eine NN-Sequenz der Länge n und durchschnittlicher Baumtiefe t mit $\beta \cdot (\alpha \cdot t) \cdot n$ Unifikationen mit Mustern beschrieben werden, wenn α einen konstanten Indizierungsaufwand und β die durchschnittliche Komplexität für die Unifikationen beschreibt. Abstrahiert man vom evtl. exponentiellen Aufwand für Unifikation (was in bezug auf realistische Programme keine echte Einschränkung ist), so ist die Komplexität größenordnungsmäßig linear in der Länge der Ableitung.

Allgemeiner wird das Narrowing durchgeführt, bis eine Kopfnormalform errechnet ist. Dann sind OINC und NN allerdings nicht mehr direkt vergleichbar, weil eine OINC-Ableitung zur leeren Sequenz $\Box$ dann unmöglich ist.

Die angegebene Strategie kann nun noch verbessert werden.
Wie bereits erwähnt, ermöglicht es eine geschickte Indizierungsstrategie (die nicht schwierig zu implementieren ist), die Verzweigung in einen Unterbaum mit konstantem Aufwand durchzuführen (sofern die Position Toplevel eines der Funktionsargumente ist, s.u.). Dazu wird das Argument der Verzweigungsposition indiziert. Der konstante Aufwand ergibt sich aus der Tatsache, daß die WAM-Instruktionen `switch_on_constant` und `switch_on_structure` normalerweise mittels einer Hashtabelle implementiert werden ([Han92]).

Die Indizierung wird schwierig, wenn die Verzweigungsposition eines Knotens unterhalb des Toplevels eines Funktionsarguments liegt. Außerdem ist es in LR-DTs gewissermaßen unnötig, die Unifikation der Formalparameter des Prädikats mit den Aktualparametern (der auszuwertende Term) links der Verzweigungsposition durchzuführen.
Diesen zwei Sachverhalten liegt die folgende Gemeinsamkeit zugrunde: Links oder über der Verzweigungsposition muß "nichts mehr unifiziert" werden, weil das bereits geschehen ist. Musterpositionen rechts der Verzweigungsposition

sind nach Definition Variablen; ihre Unifikation ist also nicht mehr als das "Umhängen" von Zeigern auf dem Heap.

Das Ignorieren dieser Positionen allein ist allerdings nicht ausreichend, weil Variablenbindungen weiter links stehender Positionen relevant sind:

Beispiel 3.4.2.
In obigem Beispiel muß die Unifikation des ersten Formalparameters des Prädikats `leq_1_2`*/3 (Muster des Knotens mit Verzweigungsposition* $1 \cdot 2$*) mit dem entsprechenden Aktualparameter nicht durchgeführt werden, weil sie bereits im Prädikat* `leq_1`*/3 durchgeführt wurde. Die Variablenbindung für* `A` *wird allerdings im Rumpf von* `leq_1_2` *benötigt.*

■

Das Problem kann behoben werden, wenn die der PROLOG-Implementierung zugrundeliegende WAM Prädikate zuläßt, die "Variablen teilen". Im angegebenen Beispiel müßte beispielsweise die Variable `A` geteilt werden, d.h. die Bindung von `A` müßte bei Aufruf von `leq_1_2`/3 noch vorhanden sein. Wenn man solches geteilten Variablen zuläßt, muß berücksichtigt werden, daß die resultierende Bindung lokal für eine Aufruffolge für ein solches Prädikat ist, d.h. daß bei erneutem Aufruf des Wurzelprädikats (im Beispiel `leq`/3) frische Variablen für die Formalparameter bereitgestellt werden.

Die Bereitstellung solcher gewissermaßen globalen Variablen läuft allerdings dem PROLOG zugrundeliegende Paradigma entgegen. Deshalb soll hier auf die (triviale) Erläuterung des Beispiels verzichtet werden. Eine andere Möglichkeit besteht darin, neben einer geschickten Indizierung der Argumente (d.h. beispielsweise Umstellen der Variablen) außerdem mit einem partiellen Auswerter Konstruktorköpfe bereits unifizierter Argumente zu eliminieren, so daß insbesondere keine Unifikation mit einem Argument mehrfach erfolgen muß (im Beispiel kann man in den Regeln für `leq_1_2` den Konstruktorkopf `s` weglassen). Verfolgt man diese Strategie konsequent, so läßt sich wiederum ein nahezu konstanter Aufwand für dan Abstieg in einem DT erzielen, wenn die zugrundeliegende WAM Argumentvariablen, die vor der Optimierung einen Konstruktorkopf hatten, in den Argumentregistern beläßt.
Da die vorangegangenen Punkte eine konkrete Implementierung voraussetzen, auf die hier nicht eingegangen wird, sollen die Konsequenzen aufgezeigt werden, die sich aus einer solchen Implementierung ergeben. Dabei wird wie bisher auch ausschließlich die Komplexität zur Laufzeit betrachtet.
Der Aufwand für den Abstieg in einem definierenden Baum entspricht dann nämlich dem Aufwand (multipliziert mit einem konstanten Faktor für die Indizierung und das Binden von Variablen) *einer einzigen* Unifikation mit der anzuwendenden linken Regelseite. Bei n Narrowingschritte ergibt sich damit als Komplexität $n \cdot (\gamma \cdot t)$ für eine Konstante γ, wobei γ sehr viel kleiner als $\beta \cdot \alpha$ aus obiger Argumentation ist.

Setzt man insbesondere für t die maximale Tiefe der DTs ein, so ergibt die bisherige Argumentation das folgende Resultat:

Lemma 3.4.3.
Die (Zeit-)Komplexität von NN ist (fast) linear in der Länge der Narrowingsequenz, wenn keine exponentielle Zeit erfordernde Unifikation durchgeführt wird. Insbesondere kann der Abstieg in einem DT in nahezu konstanter Zeit bewerkstelligt werden, wenn man eine maximale Baumtiefe als konstant voraussetzt.

Auf einen formalen Beweis wird hier verzichtet.

□

Die Tiefe der Bäume, d.h. die Größe von t kann nun für realistische Programme größenordnungsmäßig mit 2 approximiert werden. Damit beträgt der Aufwand eines erfolgreichen NN-Schritts etwa den doppelten Aufwand der Unifikation mit den einzelnen anzuwendenden linken Regelseiten.

In einer OINC-Implementierung sind für eine erfolgreiche Ableitung von n Schritten n Unifikationen mit linken Regelseiten erforderlich. Auch hier kann die Zeit zum Auffinden des "passenden" Prädikats vernachlässigt werden.

Allerdings ist der Nichtdeterminismus in OINC viel gravierender, weil jede Regel gleich bei Auftreten eines Funktionssymbols auf Toplevel ausprobiert werden muß. Nichtdeterminismus (in der PROLOG-Terminologie beispielsweise das Anlegen von Backtrackpunkten) ist aber eine sehr teure Operation, die den scheinbaren Komplexitätsvorteil von OINC um den geschätzten Faktor 2 mehr als kompensiert, zumal daraus die Existenz vieler schließlich fehlschlagender Berechnungen folgt.
Als Beispiel sei nocheinmal das Narrowing auf Grundtermen genannt, für das NN erwiesenermaßen optimal, d.h. deterministisch, ist.
Es kann also davon ausgegangen werden, daß sich NN in der Praxis sehr viel effizienter verhält. Als Beispiel seien die folgenden experimentellen Resultate angegeben, wobei als Implementierung für NN die oben angegebenen Klauseln verwendet wurden. Die Implementierung von OINC ist in Kapitel 5 beschrieben.

3.4.2 Experimentelle Resultate

Die in Tab. 1 angegebenen Zeiten wurden auf einem Intel Pentium mit 133 MHz bei 32MB Hauptspeicher unter Linux gemessen (Angabe in Sekunden). Die Implementierung erfolgte in Sicstus Prolog 3.5; die Programme wurden compiliert. Die Berechnung stoppt nach der ersten erfolgreichen Ableitung. Da der verwendete NN-Implementierung compilierte Programme zugrundeliegen und OINC als Interpreter realisiert wurde, ist der Vergleich der Zeiten allein natürlich unfair. Deshalb wird auch die Anzahl der notwendigen Choice-Points berücksichtigt.
Für die Benchmarks wurden die Zahlen in $s/0$-Schreibweise umgewandelt und kein term-sharing verwendet. Für NN kann das sehr elegant implementiert wer-

den (vgl. das zitierte Papier); eine Übertragung auf OINC ist mit derselben Strategie möglich.

Nr.	Ziel[1]	NN	OINC	Verhältnis
1	$10000 + 10000 \equiv 20000 \approx \top$	0.13	7.71	1 : 59
1a	$10000 + 10000 \approx 20000$	–	1.49	–
2	$10000 \leq 10000 + 10000 \equiv \top \approx \top$	0.14	3.93	1 : 28
3	$1000 \leq x + x \equiv \top \approx \top$	1.25	23.81	1 : 19
4	$400 + x \leq (x + 200) + x \equiv \top \approx \top$	1.03	34.9	1 : 34
5	$2000 \leq 1000 + (x + x) \equiv \top \approx \top$	1.26	24.26	1 : 19
6	$2000 + x \leq 1000 + y \equiv \top \approx \top$	0.04	1.17	1 : 29

Tab.1: Implementierungen von OINC und NN im Vergleich

Für das Ziel 1a wurde keine Zeit für NN angegeben, weil hier mit der OINC-Gleichheit $\approx$ gerechnet wird.
Der enorme Geschwindigkeitsvorteil von NN liegt neben der Tatsache, daß die Regeln für $\leq$ und $+$ in PROLOG-Prädikate *compiliert* wurden, in der hohen Anzahl an Choice-Points, die im Verlauf einer OINC-Berechnung angelegt werden, sowie die daraus resultierende höhere Anzahl fehlschlagender Berechnungen. Tabelle 2 gibt einen Überblick (Angaben in Tausend). `solve` ist das Prädikat, das den Kalkül realisiert, `getRule` wählt nichtdeterministisch eine passende Regel, und `createEq` erzeugt die neuen Gleichungen im Fall von [on]- oder [d]-Schritten (für dieses Prädikat ist auch die Anzahl der Aufrufe, wiederum in Tausend, angegeben). Die gemessenen Zeiten sind virtuelle (aber skalierbare) Zeiten, s. [SIC96]. Die Ziele sind dieselben wie in Tabelle 1, die Gesamtzeit (letzte Spalte) gibt die insgesamt benötigte Zeit (d.h. bis $\Box$ abgeleitet wurde) an, wobei Prädikate wie `append` nicht berücksichtigt wurden. Deshalb besteht auch kein proportionaler Zusammenhang zu den Zeiten in Tabelle 1.

	`solve`		`getRule`		`createEq`		
Ziel	CP[2]	Time[3]	CP	Time	Calls	Time	Σ Time[4]
1	70	69	10	5	120	23	97
1a	360	358	60	36	640	127	521
2	190	180	30	17	310	61	258
3	1138	1096	127	86	1893	371	1553
4	1710	1616	293	155	2769	547	2318
5	1157	1114	130	88	1924	377	1579
6	58	55	10	5	92	18	78

Tab.2: Profil der OINC-Berechnungen aus Tab. 1[5]

[1] $\top$ steht für *true*; die rechten Seiten entfallen für NN
[2] Choice Points
[3] virtuelle Zeit
[4] virtuelle Gesamtzeit
[5] alle Angaben in Tausend, gerundet

Auffällig ist neben der hohen Anzahl an Choice-Points die Tatsache, daß das Erzeugen neuer Gleichungen durchschnittlich 24% der Gesamtzeit (bei sehr kleiner Varianz) in Anspruch nimmt – ein Vorgang, der in NN nicht durchgeführt werden muß.
Für realistischere (d.h. kleinere) Zahlenwerte ergibt sich ein ähnliches Bild. Tabelle 3 zeigt die Profile für ähnliche Ziele wie in Tabelle 1, nur mit kleineren Zahlenwerten.

	`solve`		`getRule`		`createEq`		
Ziel[6]	CP	Time	CP	Time	Calls	Time	Σ Time
$1+1 \equiv 2$	56	55	11	3	98	18	76
$5 \leq 5+5$	102	96	17	5	164	31	132
$3 \leq x+x$	78	69	12	2	105	20	91
$4+x \leq (x+2)+x$	418	386	73	34	622	121	541
$20 \leq 10+(x+x)$	458	427	66	35	705	137	599
$20+x \leq 10+y$	597	561	105	51	944	186	798

Tab.3: Profile mit "realistischeren" Zahlen

Auch für diese Auswertungen beträgt das Verhältnis der Zeit für die Erzeugung neuer Gleichungen zur Gesamtzeit etwa ein Viertel (23%).
Für einen direkten Vergleich werden nun abschließend in Tabelle 4 die Zahlen für NN und OINC gegenübergestellt:

	NN		OINC	
Ziel	CP	Time	CP	Time
$10000+10000 \equiv 20000$	0	18480	420025	521346
$10000 \leq 10000+10000$	10000	18980	220005	258102
$1000 \leq x+x$	127252	162260	1265016	1552689
$400+x \leq (x+200)+x$	61102	133947	2003356	2317913
$2000 \leq 1000+(x+x)$	128252	164158	1287025	1578519
$2000+x \leq 1000+y$	3002	5023	68018	78103
$1+1 \equiv 2$	0	1	67	76
$5 \leq 5+5$	5	4	119	132
$3 \leq x+x$	12	4	90	91
$4+x \leq (x+2)+x$	19	28	491	541
$20 \leq 10+(x+x)$	47	52	524	599
$20+x \leq 10+y$	32	46	702	798

Tab.4: Vergleich der Anzahl der Choice-Points

Auffallend ist insbesondere, daß es im ersten und siebten Fall keine Choice-Points gibt. Das liegt an der bereits angesprochenen Eigenschaft, deterministisch auf Grundtermen zu sein. Im zweiten und achten Fall läßt sich die

[6] ohne rechte Seiten

Anzahl der Choice-Points ebenfalls auf Null reduzieren, wenn man das Prädikat `leq_1_2` über das zweite Argument indiziert. Im angegebenen, nicht optimierten PROLOG-Programm werden Choice-Points bei Aufruf der Prädikate `leq_1_2(s(_),null)` bzw. `leq_1_2(s(A),s(B))` angelegt.
Verallgemeinert man diese Argumentation, so ergibt sich sofort, daß bei der Bearbeitung von Grundtermen *keine* Choice-Points angelegt werden müssen.
Tabelle 5 gibt der Vollständigkeit halber einen Überblick bei optimiertem Code (kein term-sharing):

Ziel	NN CP	NN Time	Verhältnis CP NN:OINC
$10000 + 10000 \equiv 20000$	0	13700	1:∞
$10000 \leq 10000 + 10000$	0	10020	1:∞
$1000 \leq x + x$	1002	79654	1:1262
$400 + x \leq (x + 200) + x$	402	71175	1:4983
$2000 \leq 1000 + (x + x)$	1002	80656	1:1284
$2000 + x \leq 1000 + y$	2002	3181	1:34

Tab.5: optimierter Code für NN; kein term-sharing

Zusammenfassend läßt sich also festhalten:

1. Nimmt man die Tiefe der DTs als konstant an, so ist die Zeit, die NN für einen vollständigen Abstieg benötigt, ungefähr proportional zu der Zeit, die die Unifikation mit einer linken Regelseite benötigt.

2. Werden strikte Gleichungen betrachtet, so interessiert man sich für die Reduktion auf Konstruktorterme. Dann ist die Komplexität von NN für die Berechnung einer Lösung ungefähr proportional zum Aufwand der Unifikation mit den angewendeten linken Regelseiten. Interessiert man sich für die Reduktion auf allgemeine Kopfnormalformen, so geht die Vergleichbarkeit mit OINC verloren (keine erfolgreichen Ableitungen zu $\Box$).

3. NN kann effizient implementiert werden. Insbesondere ist die Anzahl der Choice-Points für die Reduktion von Grundtermen gleich Null.

4. OINC's Verhalten ist sehr viel schlechter. Das liegt daran, daß Regeln gleich zu Beginn ausprobiert werden und deshalb viele Choice-Points angelegt werden müssen, woraus die Existenz einer höheren Anzahl fehlschlagender Berechnungen resultiert. Die verwendete Implementierung von NN ist nicht optimiert. [Han95] schlägt mehrere Strategien vor: Partielle Auswertung der Aktualparameter bei Compilierung, geschickte Indizierung und term-sharing.

5. OINC's ungünstiges Verhalten resultiert auch aus der Tatsache, daß die Erzeugung neuer Gleichungen sehr zeitaufwendig ist.

3.5 Simulation von s-OINC und NN

Wie für NN und OINC ist es möglich, ein gegenseitiges Simulationsresultat für NN und s-OINC (Kap. 2.4) zu erhalten.
In s-OINC werden keine Regeln für die strikte Gleichheit angewendet (das Symbol $\equiv$ befindet sich üblicherweise auf Toplevel oder direkt unter einem $\wedge$), sondern mit neuen Inferenzregeln behandelt. Diese sind, wie sich im folgenden zeigen wird, [ims] und [ds] (und [ts], aber mit dieser Regel werden die Lösungen beider Kalküle unvergleichbar).
NN berechnet bei Verwendung von LR-DTs strikte Gleichungen so, daß erst die linke Seite zu einem Konstruktorkopfterm oder einer Variable abgeleitet wird, und danach die rechte. s-OINC implementiert dieses Verhalten explizit.
Die beiden Gleichungsseiten können auch in s-OINC bzw. OINC *unabhängig* voneinander zu Konstruktorkopftermen oder Variablen abgeleitet werden, weil unterhalb des berechneten Konstruktorkopfes noch kein Narrowing durchgeführt wurde und deshalb insbesondere keine Restgleichungen entstehen, die evtl. später behandelt werden müssen.
Es sei hier erneut darauf hingewiesen, daß die s-OINC zugrundeliegenden TES keine Regeln für die strikte Gleichheit enthalten müssen. Wenn sie allerdings auf rechten Regelseiten auftaucht, dann müssen die Regeln in das TES aufgenommen werden.

Da die [on]- und [ons]-Inferenzen nicht völlig unabhängig voneinander sind ([ons]-Inferenzen erzeugen sowohl Gleichungen der Form $s \equiv t$, als auch $s \approx t$), muß der Begriff der [ons]-Lösung eingeführt werden (in Analogie zur [on]-Lösung). Mit diesem Begriff ist es später möglich, Aussagen über NN-Schritten entsprechende [ons]-Schritte zu treffen.
s-OINC soll hier nicht zu s-OINC' erweitert werden, weil der Formalismus an sich keine neuen Erkenntnisse birgt. So, wie in OINC ein [on]-Schritt als gelöst bezeichnet wird, wenn eine Instanz der angewendeten rechten Regelseite zuerst als linke Seite der linkesten Gleichung der inferierten Gleichungsmenge auftaucht, so wird ab jetzt ein [ons]-Schritt als gelöst bezeichnet, wenn eine Instanz der angewendeten rechten Regelseite zuerst als linke oder rechte Seite einer strikten Gleichung auftaucht, die die linkeste Gleichung einer inferierten Gleichungsmenge ist.

Beispiel 3.5.1. *([ons]-Lösung)*
Für das TES $\mathcal{R} = \{g(a) \to a, f(x) \to h(x), h(x) \to b\}$ und die initiale Gleichung $f(g(x)) \equiv b$ gibt es die s-OINC-Ableitung

$$\begin{array}{lll} f(g(x)) \equiv b & \rightsquigarrow_{[ons]} & g(x) \approx x', h(x') \equiv b \\ & \rightsquigarrow_{[on]} & x \approx a, a \approx x', h(x') \equiv b \\ & \rightsquigarrow^2_{\{x \mapsto a, x' \mapsto a\}} & \underline{h(a) \equiv b} \\ & \rightsquigarrow_{[ons]} & a \approx x'', b \equiv b \\ & \rightsquigarrow_{\{x' \mapsto a\}} & \underline{b \equiv b} \\ & \rightsquigarrow_{[ds]} & \Box. \end{array}$$

Dabei ist die erste unterstrichene Gleichung [ons]-gelöst bzgl. des ersten [ons]-Schritts und die zweite unterstrichene Gleichung [ons]-gelöst bzgl. des zweiten [ons]-Schritts.

■

Damit kann nun eine NN-Sequenz durch eine s-OINC-Sequenz simuliert werden. In der Formulierung des Satzes bezeichnen die e_i strikte Gleichungen (oder strikte Gleichungsmengen), bei denen beide Seiten Variablen oder Konstruktorkopfterme sind. NN und s-OINC erzeugen diese Gleichungen in derselben Reihenfolge; für die dazwischenliegenden Schritte gelten dieselben Betrachtungen wie für die Simulation von NN durch OINC.

Satz 3.5.2.

- *Wenn es eine NN-Sequenz* $\overbrace{s \equiv t}^{e_0} \leadsto^* e_1 \leadsto^* e_2 \leadsto^* \cdots \leadsto^* e_k$ *mit Lösung* σ *gibt, die schließlich zu true abgeleitet wird, wobei*

  1. $e_i = s_{i,1} \equiv t_{i,1} \wedge \ldots \wedge s_{i,n_i} \equiv t_{i,n_i}$ *mit* $Head(s_{i,j}) \in \mathcal{C} \cup \mathcal{V}ar$ *und* $Head(t_{i,j}) \in \mathcal{C} \cup \mathcal{V}ar$ *für* $0 \leq i \leq k$ *und* $0 \leq j \leq n_i$ *ist,*

  2. $s_{i,j} \neq true$ *und* $t_{i,j} \neq true$ *für* $0 \leq i \leq k$ *und* $0 \leq j \leq n_i$ *ist (Gleichungen der Form* $true \equiv true \wedge \ldots$ *wurden dann direkt vorher reduziert) und*

  3. *in jeder Teilsequenz* $e_j \leadsto^* e_{j+1}$ *Narrowing außer im ersten Schritt nur an Positionen stattfindet, an denen kein* $\equiv$ *steht,*

  dann gibt es eine s-OINC-Sequenz (ohne die [ts]-Inferenzregel) $e'_0 \leadsto^* e'_1 \leadsto^* e'_2 \leadsto^* \cdots \leadsto^* e'_k$ *mit derselben Lösung* σ *(modulo Variablenumbenennung), die schließlich zu* $\Box$ *abgeleitet wird, wobei*

  1. *für* $0 \leq i \leq k$ *die* e'_i *gleich den* e_i *sind, wenn die* $\wedge$ *durch Kommata ersetzt werden,*
  2. *die [on]- und [ons]-Schritte in derselben Reihenfolge gelöst wie die entsprechenden NN-Schritte in der NN-Sequenz durchgeführt werden,*
  3. *für die Teilsequenzen* $e_j \leadsto e_{j+1}$ *die Simulationsbeziehung wie in Satz 3.3.11 gilt.*

- *Die Anzahl der NN-Schritte ist gleich der Anzahl der [on]-, [ons]-, [ds]- und [ims]-Inferenzen in der s-OINC-Sequenz abzüglich der Anzahl der Schritte für die Elimination von trues. NN-Narrowingschritte an Positionen, deren Symbol* $\equiv$ *ist, werden durch [ims]- oder [ds]-Schritte simuliert.*

Beweis
durch vollständige Induktion über die Länge der NN-Ableitung.

Induktionsverankerung durch Fallunterscheidung über s:
1. Fall: $Head(s) \in \mathcal{F}$
Eine NN-Sequenz für eine strikte Gleichung hat – LR-DTs vorausgesetzt – stets die Form

$$s \equiv t \rightsquigarrow^* \overline{s} \equiv \sigma(t) \rightsquigarrow^* \psi(\overline{s}) \equiv \overline{t} \overbrace{\rightsquigarrow \psi(\overline{s})|_1 \equiv \overline{t}|_1 \wedge \ldots \wedge \psi(\overline{s})|_k \equiv \overline{t}|_k}^{\text{falls } \psi(\overline{s}) \notin \mathcal{V}ar \text{ und } \overline{t} \notin \mathcal{V}ar},$$

wobei $\overline{s}$ und $\overline{t}$ Terme sind, die Variablen sind oder auf Toplevel ein Konstruktorsymbol haben (wenn die Ableitung erfolgreich ist, d.h. schließlich zu *true* abgeleitet wird, ist dieses Konstruktorkopfsymbol auf beiden Seiten identisch, und wenn es die Stelligkeit n hat, folgt daraus die Existenz des letzten Schritts). Dabei ist σ die Substitution, die zur Errechnung von $\overline{s}$ geführt hat und ψ die Substitution, die zur Errechnung von $\overline{t}$ geführt hat, d.h. $\sigma(t) \rightsquigarrow^*_{\psi} \overline{t}$.
NN leitet also zunächst die linke strikte Gleichungsseite zu einem Konstruktorkopfterm (oder einer Variablen) und dann die rechte zu einem Term mit demselben Kopfsymbol (oder einer Variablen) ab.

s-OINC verhält sich identisch: Für die Ableitung der linken Gleichungsseite ergibt sich die Inferenz

$$\begin{array}{lll}
s \equiv t & \rightsquigarrow_{[ons]} & s|_1 \approx l_1|_1, \ldots, s|_{n_1} \approx l_1|_{n_1}, r_1 \equiv t \\
& \rightsquigarrow^* & \sigma_1(r) \equiv \sigma_1(t) \\
& \rightsquigarrow_{[ons]} & \sigma_1(r_1)|_1 \approx l_2|_1, \ldots, \sigma_1(r_1)|_{n_2} \approx l_2|_{n_2}, r_2 \equiv \sigma_1(t) \\
& \rightsquigarrow^* & \sigma_2(r_2) \equiv \sigma_2(\sigma_1(t)) \\
& \rightsquigarrow_{[ons]} & \sigma_2(r_2)|_1 \approx l_3|_1, \ldots, \sigma_2(r_2)|_{n_3} \approx l_3|_{n_3}, r_3 \equiv \sigma_2(\sigma_1(t)) \\
& \rightsquigarrow^* & \sigma_3(r_3) \equiv \sigma_3(\sigma_2(\sigma_1(t))) \\
& & \vdots \\
& \rightsquigarrow & \overline{s} \equiv \sigma'(t),
\end{array}$$

wobei solange [ons]-Inferenzen auf Instanzen rechter Regelseiten angewendet werden, bis dort ein Konstruktorkopfterm oder eine Variable steht ($l_i \to r_i$ sind Regeln mit $l_i = f_i(l_i|_1, \ldots, l_i|_{n_i})$ für $f_i = Head(r_{i-1}) \in \mathcal{F}$ mit $Arity(f_i) = n_i$ für $i \geq 1$).

Diese Sequenz kann nun durch eine OINC-Ableitung simuliert werden.
Für einen Konstruktor c der Stelligkeit k mit $\overline{s} = c(\overline{s}|_1, \ldots, \overline{s}|_k)$ und (frische) Variablen $x_1, \ldots, x_k$ (abgekürzt durch $\vec{x_k}$) und $y_1, \ldots, y_k$ (abgekürzt durch $\vec{y_k}$) ergibt sich unter Anwendung derselben Regeln vom zweiten Schritt an die Inferenzfolge bzgl. $\mathcal{R}^{\equiv}$:

$$
\begin{array}{lll}
s \equiv t \approx true & \rightsquigarrow_{[on]} & s \approx c(\vec{x_k}), \overbrace{t \approx c(\vec{y_k}), x_1 \equiv y_1 \wedge \ldots \wedge x_k \equiv y_k \approx true}^{E} \\
& \rightsquigarrow_{[on]} & s|_1 \approx l_1|_1, \ldots, s|_{n_1} \approx l_1|_{n_1}, r_1 \approx c(\vec{x_k}), E \\
& \rightsquigarrow^* & \sigma_1(r_1) \approx c(\vec{x_k}), \sigma_1(E) \\
& \rightsquigarrow_{[on]} & \sigma_1(r_1)|_1 \approx l_2|_1, \ldots, \sigma_1(r_1)|_{n_2} \approx l_2|_{n_2}, r_2 \approx c(\vec{x_k}), \sigma_1(E) \\
& \rightsquigarrow^* & \sigma_2(r_2) \approx c(\vec{x_k}), \sigma_2(\sigma_1(E)) \\
& \rightsquigarrow_{[on]} & \sigma_2(r_2)|_1 \approx l_3|_1, \ldots, \sigma_2(r_2)|_{n_3} \approx l_3|_{n_3}, r_3 \approx c(\vec{x_k}), \\
& & \sigma_2(\sigma_1(E)) \\
& \rightsquigarrow^* & \sigma_3(r_3) \approx c(\vec{x_k}), \sigma_3(\sigma_2(\sigma_1(E))) \\
& & \vdots \\
& \rightsquigarrow & \overline{s} \approx c(\vec{x_k}), \sigma'(E) \\
& \rightsquigarrow_{[d]} & \overline{s}|_1 \approx x_1, \ldots \overline{s}|_k \approx x_k, \sigma'(E) \\
& \rightsquigarrow^k_{[v]} & \vartheta\big(t \approx c(\vec{y_k}), \overline{s}|_1 \equiv y_1 \wedge \ldots \wedge \overline{s}|_k \equiv y_k \approx true\big),
\end{array}
$$

und der Simulationssatz 3.3.11 zeigt, daß die Anwendung der Regeln vom zweiten [on]-Schritt an den NN-Schritten entspricht und daß $\vartheta = \sigma[\mathcal{V}ar(s) \cup \mathcal{V}ar(t)]$ ist. Man beachte, daß die $\vec{x_k}$ frische Variablen sind, die nirgendwo anders in der inferierten Gleichungsmenge auftreten können. Da dieselben Regeln in derselben Reihenfolge angewendet wurden, folgt $\vartheta = \sigma'[\mathcal{V}ar(s) \cup \mathcal{V}ar(t)]$.

Die Fortsetzung der Sequenzen ergibt sich nun in Abhängigkeit des Aussehens von t:
Wenn $Head(t) = c$ ist, ergibt sich die OINC- (und damit eine entsprechende NN-)Ableitung

$$\ldots \rightsquigarrow^{k+1}_{[d,v]} \overline{s}|_1 \equiv \overline{t}|_1 \wedge \ldots \wedge \overline{s}|_k \equiv \overline{t}|_k \approx true,$$

womit bei einer Anwendung von [ds] in der s-OINC-Ableitung auf die zuletzt inferierte Gleichungsmenge der Beweis erbracht ist (auch hier vergegenwärtige man sich die Tatsache, daß die $\vec{y_k}$ frische Variablen sind, die nirgendwo sonst in der inferierten Gleichungsmenge auftreten können). Die NN-inferierte Gleichungsmenge unterscheidet sich von der OINC-inferierten durch das Fehlen der rechten $\approx$-Gleichungsseite *true*. Davon wird im folgenden abstrahiert.
Für $t \in \mathcal{V}ar$ ergibt sich die OINC- (und damit NN-)Sequenz

$$\ldots \rightsquigarrow_{[v]} \overline{s}|_1 \equiv y_1 \wedge \ldots \wedge \overline{s}|_k \equiv y|_k \approx true.$$

s-OINC generiert in diesem Fall mittels eines [ims]-Schritts eine identische Gleichungsmenge.

Wenn auch $Head(t) \in \mathcal{F}$ ist, dann gibt es eine s-OINC-Sequenz

$$
\begin{array}{lll}
\overline{s} \equiv \sigma'(t) & \rightsquigarrow_{[ons]} & \sigma'(t)|_1 \approx l'_1|_1, \ldots, \sigma'(t)|_{n'_1} \approx l'_1|_{n'_1}, \sigma'(\overline{s}) \equiv r'_1 \\
& \rightsquigarrow^* & \psi_1(\sigma'(\overline{s})) \equiv \psi_1(r'_1) \\
& & \vdots \\
& \rightsquigarrow & \psi_j \circ \cdots \circ \psi_1 \circ \sigma'(\overline{s}) \equiv \overline{t},
\end{array}
$$

für $j \in \mathbb{N}$ genau wie im vorherigen symmetrischen Fall, nur daß jetzt die [ons]-Schritte auf rechte Gleichungsseiten angewendet werden und hier Regeln $l'_i \to r'_i$ zur Anwendung kommen.

OINC (und damit NN) errechnen

$$\begin{array}{ll} & \sigma\big(t \approx c(\vec{y_k}), \overline{s}|_1 \equiv y_1 \wedge \ldots \wedge \overline{s}|_k \equiv y_k \approx true\big) \\ \rightsquigarrow^* & \psi_j \circ \cdots \psi_1 \circ \sigma(\overline{s}|_1 \equiv \overline{t}|_1 \wedge \ldots \wedge \overline{s}|_k \equiv \overline{t}|_k \approx true) \end{array}$$

mit $\psi \restriction_{\mathcal{V}ar(\overline{s}) \cup \mathcal{V}ar(\overline{t})} = \psi_j \circ \cdots \psi_1 \circ \sigma \restriction_{\mathcal{V}ar(s) \cup \mathcal{V}ar(t)}$ nach Satz 3.3.11.
Die von OINC bzw. NN inferierte Gleichung entsteht aus der von s-OINC inferierten durch Anwendung eines [ds]-Schritts. Da $\sigma = \sigma'$ (mit der üblichen Variableneinschränkung) bereits gezeigt wurde, ist damit die Behauptung gezeigt.

2. Fall: $Head(s) \in \mathcal{C}$

- Wenn $Head(t) \in \mathcal{F}$ ist, dann ist der Beweis identisch zum ersten Fall, nur daß die Ableitung von s zu $\overline{s}$ weggelassen werden kann.
- Wenn $Head(t) \in \mathcal{C}$ ist, dann muß $c := Head(s) = Head(t)$ sein, weil andernfalls keine Ableitung zu $true$ möglich ist.

  NN wendet in diesem Fall Narrowing an Position ε an, wobei an dieser Stelle $\equiv$ steht. Wenn $s = c(s_1, \ldots, s_n)$ und $t = c(t_1, \ldots, t_n)$ ist, dann entsteht die Gleichung $s_1 \equiv t_1 \wedge \ldots \wedge s_n \equiv t_n$.

  s-OINC kann dann eine [ds]-Inferenz mit Resultat $s_1 \equiv t_1, \ldots, s_n \equiv t_n$ durchführen.

  Falls $n = 0$ ist, so reduziert NN zu $true$ und s-OINC zu $\Box$.
- Wenn $t \in \mathcal{V}ar$ ist, dann führt NN einen Narrowingschritt an Position ε durch. Für $s = c(s_1, \ldots, s_n)$ und $t = x$ ergibt sich die NN-Sequenz $c(s_1, \ldots, s_n) \equiv x \rightsquigarrow_{\{x \mapsto c(x_1, \ldots, x_n)\}} s_1 \equiv x_1 \wedge \ldots \wedge s_n \equiv x_n$ für frische Variaben $x_1, \ldots, x_n$.

  s-OINC führt eine [ims]-Inferenz durch: $c(s_1, \ldots, s_n) \equiv x \rightsquigarrow_{[ims]} \vartheta(s_1 \equiv x_1, \ldots, s_n \equiv x_n)$ mit $\vartheta = \{x \mapsto c(x_1, \ldots, x_n)\}$. Wiederum sind die Ergebnisse in beiden Fällen identisch.

3. Fall: $s \in \mathcal{V}ar$

- Falls $Head(t) \in \mathcal{F}$ ist, so wird der Beweis wie im ersten Fall geführt, nur daß die Ableitung von s zu $\overline{s}$ nicht betrachtet werden muß.
- Falls $Head(t) \in \mathcal{C}$ ist, so wird der Beweis wie im zweiten Fall geführt. Dabei wendet s-OINC die symmetrische [ims]-Inferenz an; das Ergebnis ist wiederum identisch (mit in bezug auf Fall 2 vertauschten linken und rechten strikten Regelseiten).

- Falls auch $t \in \mathcal{V}ar$ ist, so führt NN Narrowing mit allen Regeln für strikte Gleichungen durch. Um vergleichbare Lösungen zu erhalten, muß auch s-OINC ein ähnliches Verfahren anwenden. s-OINC wendet in diesem Fall eine [ts]-Inferenz an und kann so allgemeinere Lösungen ermitteln. Damit können solche Ableitungen nicht mit NN verglichen werden, es sei denn, auch NN erlaubt solche strikten Variablenbindungen, oder man ersetzt [ts] durch eine Inferenzregel, die die beiden Variablen an alle vorhandenen Konstruktoren bindet (s. dazu auch das Kapitel über den Vergleich der Komplexitäten und die Implementierung beider Kalküle).

Die Behauptung über die Anzahl der Schritte ergibt sich direkt aus der Konstruktion. Dem OINC- (und damit NN-) Narrowing an der $\equiv$-Position entspricht jeweils eine abschließende [ims]- oder [ds]-Inferenz von s-OINC.

Induktionsschritt:
Der Beweis ist identisch zum Beweis der Induktionsverankerung. Der einzige Unterschied besteht darin, daß NN jetzt ggf. Gleichungen der Form $true \wedge s_1 \equiv t_1 \wedge \ldots \wedge s_n \equiv t_n$ berechnet, die dann in einem weiteren Schritt zu $s_1 \equiv t_1 \wedge \ldots \wedge s_n \equiv t_n$ reduziert werden. Diese Schritte haben kein Analogon in s-OINC. Da die in den Teilsequenzen berechnete Variablenbindungen identisch sind, ist insbesondere auch die resultierende Lösung identisch (bis auf Variablenumbenennung).

Die Aussagen über die Anwendungsreihenfolge der Regeln ergeben sich direkt aus Satz 3.3.35. Wesentlich ist, daß linke und rechte Seiten strikter Gleichungen hintereinander berechnet werden, bis auf beiden Seiten ein Term aus $\mathcal{T}(\mathcal{C}, \mathcal{X})$ steht. Dann werden in s-OINC [ims]- oder [ds]-Inferenzen durchgeführt, die in NN dem Narrowing mit Regeln für die strikten Gleichheit entsprechen.
Die Aussage über die Anzahl der Schritte ist unter Berücksichtigung der Bemerkung im Induktionsschritt offensichtlich.

□

In Kapitel 2 wurde angemerkt, daß s-OINC eine Lösung mehrfach berechnet, wenn bei gleichzeitiger Anwendbarkeit der [ons]-Regeln auch beide ausprobiert werden. Aus dem Beweis von Satz 3.5.2 folgt nun, daß es in einem solchen Fall ausreichend ist, sich auf die erste Variante zu beschränken.

Lemma 3.5.3.
Bei gleichzeitiger Anwendbarkeit der symmetrischen [ons]-Regeln ist es in induktiv-sequentiellen TES ausreichend, nur die erste Variante ([ons]-Schritt für die linke Gleichungsseite) zu berücksichtigen, ohne die Vollständigkeit zu verlieren.

Beweis
Im Beweis von Satz 3.5.2 wurde bei gleichzeitiger Anwendbarkeit stets die [ons]-Inferenz für die linke Gleichungsseite gewählt (und die Regel für die rechte

Gleichungsseite nach Ableiten der ersten zu einem Konstruktorkopfterm oder einer Variablen). Da NN vollständig ist, werden auf diese Weise alle Lösungen gefunden.

□

Wählt man statt der Regel [ts] eine Inferenzregel, die beide Variablen mit allen Konstruktoren instantiiert, so ergibt sich aus der Tatsache, daß bereits OINC allgemeinste Lösungen und damit dieselbe Lösungsmenge berechnet, auch die wechselseitige Simulation:

Satz 3.5.4.
Jeder erfolgreichen NN-Ableitung kann eineindeutig eine s-OINC-Ableitung zugeordnet werden. Die Anwendungsreihenfolge der Regeln in NN entspricht der Lösungsreihenfolge entsprechender Inferenzen in s-OINC. Die Anzahl der Schritte ist dieselbe, wenn die Elimination von trues in NN nicht mitgezählt wird, die keine direkte Entsprechung in s-OINC hat.

Beweis
Der Beweis ist identisch zu dem Beweis von Satz 3.3.35. Zwischen den einzelnen Regeln von s-OINC gibt es bis auf den Fall der symmetrischen [ons]-Regeln keinen Nichtdeterminismus. Berücksichtigt man Lemma 3.5.3 und codiert diesen Sachverhalt als zusätzliche Bedingung in die [ons]-Inferenzregel hinein, so können – da in bezug auf OINC kein zusätzlicher Nichtdeterminismus entsteht – keine Lösungen doppelt berechnet werden.

□

3.5.1 Vergleich der Komplexitäten: Grundsätzliches

Die Tatsache, daß die von s-OINC errechneten Ableitungen bei Anwesenheit strikter Gleichungen im Normalfall kürzer als die von OINC berechneten sind, bedeutet nicht unbedingt, daß s-OINC's Rechenverhalten auch "besser" als das von OINC ist. Das wird nur dann der Fall sein, wenn verhältnismäßig viele strikte Gleichungen erzeugt werden. Ist das nicht der Fall, so kann sich der aus der Implementierung der Inferenzregeln für die strikte Gleichheit ergebende Overhead in bezug auf die Anzahl der Choice-Points sogar negativ auswirken. Wenn allerdings eine hohe Anzahl strikter Gleichungen erzeugt wird, wird sich s-OINC sehr viel besser verhalten. Das liegt dann daran, daß die (fast immer fehlschlagende) Auswahl einer Regel für die strikte Gleichheit durch die Regeln [ims], [ds] und [ts] zeitlich nach hinten verschoben wird – die Regeln finden erst Anwendung, wenn beide Seiten Konstruktorkopfterme oder Variablen sind.
Satz 3.5.2 zeigt, daß sich s-OINC außer in einem Fall (eine Gleichungsseite ist eine Variable, die andere ein Konstruktorkopfterm; [ims]-Inferenzen) in bezug auf Nichtdeterminismus nicht besser verhält als NN.

Der Nichtdeterminismus hat in diesem Fall allerdings nicht besonders schwerwiegende Konsequenzen, weil er direkt im nächsten Schritt zum Erfolg oder Mißerfolg führt. In einer Implementierung müssen dennoch z.B. Choice-Points angelegt werden, was sehr aufwendig ist. In der weiter oben zitierten Implementierung von NN tritt dieser ungünstige Fall allerdings nur dann auf, wenn die *linke* Seite einer strikten Gleichung variabel ist; im anderen Fall wird durch Indizierung über das erste Argument die "richtige" Regel mit konstantem Aufwand ohne Anlegen von Choice-Points gefunden.
In diese Fall verhält sich also s-OINC echt "besser", was allerdings

1. evtl. durch das schlechte Verhalten in bezug auf den Nichtdeterminismus bzgl. der nach wie vor vorhandenen [on]-Inferenzen kompensiert wird und

2. auch in NN vermieden werden kann, wenn man Regeln für die strikte Gleichheit in den Kalkül hineincodiert (s.u.). Die genauere Betrachtung (incl. Beweis für Vollständigkeit und Korrektheit) sprengt allerdings den Rahmen dieser Arbeit.

Da eine Implementierung von NN für den praktischen Einsatz sinnvollerweise ebenfalls die Behandlung strikter Gleichungen in den Kalkül hineincodiert (was z.B. in der Programmierrsprache Curry der Fall ist), so wird sich NN immer noch sehr viel besser verhalten als s-OINC.
Auch s-OINC krankt an dem Problem, daß das Anlegen neuer Gleichungen sehr zeitaufwendig ist (und etwa ein Viertel der Gesamtzeit beansprucht). Außerdem werden im Normalfall in der Summe mehr nicht-strikte als strikte Gleichungen gelöst werden müssen, woraus sich wiederum die Problematik des Overheads für die Implementierung der neuen Inferenzregeln ergibt.

3.5.2 Vergleich der Komplexitäten: Experimentelle Resultate

In Tabelle 6 sind zur Veranschaulichung experimentelle Resultate angegeben.
Die Zahlen decken sich mit den getroffenen Feststellungen: Nur für die ersten zwei Ziele ist s-OINC's Verhalten in bezug auf die Anzahl der Choice-Points günstiger (Faktor 4), denn nur hier werden neue strikte Gleichungen erzeugt.

Als Fazit ergibt sich, daß s-OINC nicht ohne genaue Analyse der Programme und der möglichen Anfragen verwendet werden sollte. Wenn man durch Analyse der Ziele vorhersagen kann, ob viele neue strikte Gleichungen erzeugt werden, dann lohnt der Einsatz von s-OINC. Die Zahlen für NN beziehen sich auf nicht-optimierten Code.

Ziel[7]	s-OINC CP	s-OINC Time	OINC CP	OINC Time	NN CP
$10000 + 10000 \equiv 20000$	100K	135K	420K	521K	0
$2000 + x \equiv y$	16K	22K	64K	77K	2K
$10000 \leq 10000 + 10000 \equiv \top$	220K	306K	220K	258K	10K
$1000 \leq x + x \equiv \top$	1515K	1954K	1265K	1553K	127K
$400 + x \leq (x + 200) + x \equiv \top$	2123K	2947K	2003K	2318K	61K
$2000 \leq 1000 + (x + x) \equiv \top$	1537K	1985K	1287K	1579K	128K
$2000 + x \leq 1000 + y \equiv \top$	69K	94K	68K	78K	3K
$1 + 1 \equiv 2$	16	14	67	76	0
$5 \leq 5 + 5 \equiv \top$	116	149	119	132	5
$3 \leq x + x \equiv \top$	91	101	90	91	12
$4 + x \leq (x + 2) + x \equiv \top$	500	662	491	541	19
$20 \leq 10 + (x + x) \equiv \top$	546	712	524	599	47
$20 + x \leq 10 + y \equiv \top$	709	956	702	798	32

Tab.6: Vergleich der Anzahl der Choice-Points bei strikter Gleichheit[8]

3.6 Diskussion

Thema dieses Kapitels war eine eingehende Unzersuchung des Zusammenhangs zwischen OINC und NN bzw. s-OINC und NN. Es wurde gezeigt, wie jede NN-Ableitung durch eine OINC-Ableitung simuliert werden kann. Die Tatsache, daß die von OINC und NN errechneten Lösungen identisch sind, führte dann zu der Aussage über die Simulationsbeziehung beider Kakūle.

Diese gegenseitige Simulationsbeziehung (erfolgreiche Ableitungen) bedeutet nun aber nicht, daß beide Kalküle im praktischen Einsatz gleich gut geeignet wären. Eine Analyse der Algorithmen ergab, daß OINC's bzw. s-OINC's Design insofern schlecht gegenüber dem von NN abschneidet, als

- der bei der Auswahl der Regeln auftretende Nichtdeterminismus "zu früh" auftritt und damit mehr Regeln als in NN ausprobiert werden müssen, woraus die Existenz einer höheren Anzahl schließlich fehlschlagender Berechnungen resultiert und
- zeitaufwendig neue Gleichungen erzeugt werden müssen.

Auch wenn die Implementierung von NN sehr viel sorgfältiger stattfand und die gemessenen Werte insofern relativiert werden müssen (man beachte, daß die angegebenen Werte immer ohne die bekannten Optimierungen für NN ermittelt wurden), so spricht dieser zentrale Punkt für den Einsatz von NN bei der Verwendung induktiv-sequentieller TES (vgl. dazu Abschnitt 4.8 und die abschließende Diskussion, Kapitel 6).

[7] $\top$ für true

[8] K steht für 1000 (Angaben gerundet)

Außerdem wurde gezeigt, daß der Einsatz von s-OINC zwar zu kürzeren Ableitungen, aber nicht unbedingt zu “schnelleren” Berechnungen führt. Der Nachweis, daß bei einer geschickteren Implementierung als der hier angegebenen s-OINC's Verhalten “konkurrenzfähig” ist, ist durch Angabe einer solchen noch zu erbringen.

Als Fazit ist festzuhalten, daß bei der Verwendung induktiv-sequentieller Programme dem Kalkül NN bei der Implementierung einer funktional-logischen Sprache der Vorzug zu geben ist, zumal ein integrierendes Paradigma (von Narrowing und Residuation, [Han97]) auf der NN zugrundeliegenden Datenstruktur, den definierenden Bäumen, bereits existiert.

Interessant wäre die Untersuchung einer größeren Klasse von TES mit einem auf dieser korrekten und vollständigen Kalkül. Dazu bietet sich zunächst Parallel Narrowing [AEH97] an, das für konstruktorbasierte fast orthogonale Systeme korrekt und vollständig ist. Diese Systeme sind schwach orthogonal, und für jedes Paar $l \to r, l' \to r'$ von Regeln ist der einzige nichtvariable Unterterm von l, der mit l' unifizierbar ist, l selbst. Ein Beispiel für ein fast orthogonales System sind die Regeln für das parallele Oder. Für konstruktorbasierte TES fällt die Klasse der fast und der schwach orthogonalen TES zusammen [AEH97]; damit könnte NN mit OINC nicht mehr bzgl. induktiv-sequentieller sondern bzgl. konstruktorbasierter OTES verglichen werden. Es ist davon auszugehen, daß sich die beiden Kalküle auch bzgl. dieser Klasse von TES ähnlich verhalten (wesentlich ist dabei die Einschränkung auf orthogonale und nicht fast orthogonale Systeme). Voraussetzung dafür ist allerdings der Nachweis, daß auch die von Parallel Narrowing errechneten Lösungen unabhängig voneinander sind.

Weitere Forschungsarbeit könnte sich auch aus dem Versuch ergeben, OINC nicht durch ein Inferenzschema, sondern eine Funktion (wie die NN-Berechnungsfunktion λ) zu definieren. Damit könnten Performancegewinne durch das nicht mehr notwendige Anlegen neuer Gleichungen und ggf. eine zeitliche Verschiebung des Nichtdeterminismus bei Ausprobieren der Regeln erzielt werden.

Kapitel 4

Narrowing höherer Ordnung

In den bisherigen Abschnitten wurde Narrowing "erster Ordnung" betrachtet - "erster Ordnung" in dem Sinn, daß keine Variablen einen funktionalen Typ haben konnten. Diese Einschränkung war bisher durch die Art der verwendeten TES motiviert.
Im Kontext der deklarativen Programmierung sind solche Variablen allerdings von Interesse. Als Beispiel sei die `map`-Funktion genannt, die beispielsweise durch das folgende Curry-Programm implementiert werden kann:

```
map _ []=[]
map F [X|Xs] = [F X|map F Xs] ,
```

wobei die übliche Notation für Listen verwendet wurde. In diesem Beispiel ist `F` in der zweiten Regel eine Funktionsvariable.
Die Übersetzung in ein TES (z.B. konstruktorbasiert) gestaltet sich nun problematisch, weil der Listenkopf `F X` der rechten zweiten Regelseite kein Term im bisherigen Sinn ist. Damit kann dieses Programm insbesondere nicht ohne weiteres in ein TES transformiert werden. Außerdem ist zunächst unklar, wie eine Narrowing-Strategie mit solchen "Termen" umgehen soll: Die Behandlung von Funktionsvariablen in einer funktionalen Programmiersprache (mit der operationalen Semantik der Reduktion) ist nämlich nur ein Aspekt. Der andere, im Kontext der funktional-logischen Programmiersprachen (mit der operationalen Semantik z.B. des Narrowing), ist das Binden solcher Variablen an (partiell applizierte) Funktionen.

Im folgenden werden zwei Ansätze zur Lösung der angesprochenen Probleme diskutiert und später verglichen. Zum einen wird die implizite Applikation der Variablen `F` auf ihr Argument `X` durch eine explizite Applikation mittels eines neu eingeführten Funktionssymbols ersetzt – dieses Verfahren wird als "Warren's Methode" bezeichnet werden.
Der andere Ansatz definiert zunächst eine neue Klasse von TES, nämlich die applikativen TES und entwickelt basierend auf diesen neuen TES OINC zu einem neuen Kalkül, NCA.
Beiden Ansätzen ist gemeinsam, daß sie auf Strategien höherer Ordnung wie

z.B. die Unifikation höherer Ordnung, verzichten, und sich stattdessen auf Methoden erster Ordnung abstützen.

4.1 Warren's Methode

D.H.D. Warren diskutiert in [War82] die Notwendigkeit möglicher Erweiterungen von PROLOG um Konstrukte höherer Ordnung, nämlich Prädikatvariablen und λ-Abstraktionen. Seine Schlußfolgerung ist, daß solche Erweiterungen nichts als "mere syntactic sugar" und demzufolge nicht in den Sprachumfang von PROLOG aufzunehmen seien. Die zentrale Idee (die früher schon in [Rey72] im Kontext der funktionalen Programmierung entwickelt wurde) ist, partielle Funktionsapplikationen durch die Anwendung einer expliziten Applikationsfunktion `apply` auf neue Konstruktoren zu ersetzen und die Auswertung von `apply` durch neue Regeln zu ermöglichen. Implementierungen funktionaler Sprachen verwenden diesen Mechanismus typischerweise, um Funktionen höherer Ordnung zur Verfügung stellen zu können ("closure-Mechanismus", vgl. [BG86]).
Die Übersetzung von PROLOG-Programmen (in [CER90] wird die in [War82] fehlende explizite Konstruktion der neuen Regeln und neuen Konstruktoren beschrieben) ist im Rahmen dieser Arbeit von untergeordnetem Interesse, ebenso wie die Übertragung von λ-Ausdrücken. Es wird deshalb direkt die Übersetzung im Kontext der funktional-logischen Programmierung untersucht (vgl. [Nol95] für eine präzise Untersuchung der Methode im funktionalen Kontext; [Gon93] und [BG86] für die Übertragung in den funktional-logischen Kontext mit den Sprachen SFL [GHR92] und IDEAL [BG86]; [AT98] für eine Erweiterung: getyptes Narrowing höherer Ordnung für monomorphe Programme).

Die angedeutete Transformation eines TES in eins mit neuen Konstruktoren, die partiell applizierten Funktionen entsprechen, und einer neuen Funktion @ für die explizite Funktionsapplikation, geschieht unter Zuhilfenahmen der folgenden Definition. Dabei wird die Problematik einer Repräsentation von Termen, die Variablen als Kopfsymbole haben, aber keine Variablen sind, zunächst zurückgestellt. Außerdem werden auch Konstruktoren höherer Ordnung zugelassen. Hier wird überdies zunächst davon ausgegangen, daß auf rechten Regelseiten keine nichtvariablen Terme mit Variablen als Kopfsymbol vorkommen, was die Implementierung der `map`-Funktion (noch) unmöglich macht.

Definition 4.1.1. *($\mathcal{R}^{@}$)*
Für jedes $f \in \mathcal{F} \cup \mathcal{C}$ der Stelligkeit $n \geq 1$ werden mit Hilfe n neuer Konstruktoren $\$f, \$f(x_1), \ldots, \$f(x_1, \ldots, x_{n-1})$, die partiell applizierte Funktionen oder Konstruktoren repräsentieren, n neue Regeln definiert (AP-Gleichungen in der Terminologie von [Gon93]):

$$\begin{array}{rcl} @(\$f, x_1) & \rightarrow & \$f(x_1) \\ @(\$f(x_1), x_2) & \rightarrow & \$f(x_1, x_2) \\ & \vdots & \\ @(\$f(x_1, \ldots, x_{n-2}), x_{n-1}) & \rightarrow & \$f(x_1, \ldots, x_{n-1}) \\ @(\$f(x_1, \ldots, x_{n-1}), x_n) & \rightarrow & f(x_1, \ldots, x_n), \end{array}$$

wobei @ ein neues zweistelliges Funktionssymbol ("apply") und $x_i \in \mathcal{V}ar$ *für* $1 \leq i \leq n$ *ist. Für nullstellige Symbole werden keine Regeln eingeführt.*
Fügt man zu den Regeln eines TES $\mathcal{R}$ *für jedes Funktionssymbol diese neuen Regeln hinzu, so werde das neu entstandene TES mit* $\mathcal{R}^@$ *bezeichnet.*

△

Bemerkung 4.1.2.

1. *Die ursprünglichen Regeln werden nicht verändert. Damit verhält sich beispielsweise NN für Systeme erster Ordnung (d.h. Systemen ohne Vorkommen von @) für Anfragen erster Ordnung genau so, wie es sich bei Abwesenheit der zusätzlichen Regeln verhält.*

2. *Nullstellige Funktions- oder Konstruktorsymbole werden nicht codiert, weil nullstellige Symbole nicht partiell appliziert werden können. Auf diesen Sachverhalt wird weiter unten noch eingegangen werden.*

3. *Um Applikationen von Variablen auf Argumentterme zu ermöglichen, wird später die explizite Applikation auf rechten Regelseiten zugelassen. Das ist in bezug auf die Semantik der TES nicht unproblematisch. Darauf, sowie auf die "Korrektheit" bzw. "Vollständigkeit" der Übersetzung wird weiter unten eingegangen.*

4. *Das hier verwendete @ ist zur Vereinfachung eine polymorphe Funktion. Eigentlich müßte jede @-Regel mit ihrem Typen (d.h. der partiell applizierten Funktion bzw. dem partiell applizierten Konstruktor und der "Tiefe" der Applikation) indiziert werden. Das gleiche gilt für die neu eingeführten Konstruktoren. Darauf wird hier aus Übersichtlichkeitsgründen verzichtet; eine Optimierung des Verfahrens verwendet allerdings genau diese Typinformation ([AT98],[BBH97]).*

5. *Wenn* $\mathcal{R}$ *orthogonal ist, dann auch* $\mathcal{R}^@$ *(@ ist ein neues Funktionssymbol, und offensichtlich gibt es bei den neu eingeführten Regeln keine kritischen Paare).*

Beispiel 4.1.3.
Das TES

$$\mathcal{R} = \left\{ \begin{array}{rcl} plus(0, x) & \rightarrow & x, \\ plus(s(x), y) & \rightarrow & s(plus(x, y)), \\ inc(x) & \rightarrow & s(x) \end{array} \right\}$$

wird mit der Warren'schen Methode zu

$$\mathcal{R}^{@} = \mathcal{R} \cup \left\{ \begin{array}{rcl} @(\$plus, x_1) & \to & \$plus(x_1), \\ @(\$plus(x_1), x_2) & \to & plus(x_1, x_2), \\ @(\$inc, y_1) & \to & inc(y_1), \\ @(\$s, z_1) & \to & s(z_1) \end{array} \right\}$$

erweitert. Man beachte die Einführung einer Regel für den Konstruktor s.

■

Damit können nun Anfragen höherer Ordnung gestellt werden, wenn man das Symbol @ in Anfragen zuläßt:

Beispiel 4.1.4. *(Forstsetzung von Beispiel 4.1.3)*
Es entstehe $\mathcal{R}^{@}_{\equiv}$ aus $\mathcal{R}^{@}$ in Beispiel 4.1.3, indem man die Regeln für die strikte Gleichheit hinzunimmt (und diese auch um die neuen Regeln für neue Konstruktoren $\$\equiv, \$\equiv(x_1), \$\wedge, \$\wedge(x_1)$ erweitert). Dann gibt es für das Ziel $@(x, y) \equiv 0$ die folgende NN-Ableitung:

$$\begin{array}{rll} @(x, y) \equiv 0 & \rightsquigarrow_{\{x \mapsto \$plus(x_1)\}} & plus(x_1, y) \equiv 0 \\ & \rightsquigarrow_{\{x_1 \mapsto 0\}} & y \equiv 0 \\ & \rightsquigarrow_{\{y \mapsto 0\}} & true \end{array}$$

mit der Lösung $\{x \mapsto \$plus(0), y \mapsto 0\}$. Man beachte, daß das in der Anfrage verwendete @ nicht Element der Signatur von $\mathcal{R}$, wohl aber von $\mathcal{R}^{@}$ ist und daß das erste Argument von @ eine Variable ist.

■

Dieses Beispiel legt auch eine Interpretation der partiell applizierten Funktionen (Symbole mit einem \$ davor) als Ausdrücke des λ-Kalküls nahe: $\$plus(0)$ ist die einmal applizierte (zweistellige) Funktion *plus* mit 0 als erstem Argument, d.h. $\$plus(0) \triangleq \lambda x.plus(0, x)$ ($\triangleq$ bedeutet soviel wie "Entsprechung"). Die Regeln des λ-Kalküls müssen in das TES aufgenommen werden, wenn die Korrektheit einer Lösung nachgewiesen werden soll.

Beispiel 4.1.5.
Es sei $\mathcal{R} = \{f(a, b) \to 1, g(x) \to x\}$ ein TES. Die Anwendung der Warren'schen Methode liefert das erweiterte TES

$$\mathcal{R}^{@} = \mathcal{R} \cup \{@(\$f, x_1) \to \$f(x_1), @(\$f(x_1), x_2) \to f(x_1, x_2), @(\$g, x_1) \to g(x_1)\}$$

NN berechnet drei Lösungen für das Ziel $@(@(x, y), z) \equiv 1$: $\{x \mapsto \$f, y \mapsto a, z \mapsto b\}, \{x \mapsto \$g, y \mapsto \$f(a), z \mapsto b\}$ und $\{x \mapsto \$g, y \mapsto \$g, z \mapsto 1\}$.
Die zugehörigen Ableitungen sind

$$\begin{aligned}
@(@(x,y),z) \equiv 1 &\leadsto_{\{x \mapsto \$f\}} && @(\$f(y),z) \equiv 1 \\
&\leadsto && f(y,z) \equiv 1 \\
&\leadsto_{\{y \mapsto a, z \mapsto b\}} && 1 \equiv 1 \\
&\leadsto && \mathit{true},
\end{aligned}$$

$$\begin{aligned}
@(@(x,y),z) \equiv 1 &\leadsto_{\{x \mapsto \$g\}} && @(g(y),z) \equiv 1 \\
&\leadsto && @(y,z) \equiv 1 \\
&\leadsto_{\{y \mapsto \$f(x')\}} && f(x',z) \equiv 1 \\
&\leadsto_{\{x' \mapsto a, z \mapsto b\}} && 1 \equiv 1 \\
&\leadsto && \mathit{true} \quad \mathit{und}
\end{aligned}$$

$$\begin{aligned}
@(@(x,y),z) \equiv 1 &\leadsto_{\{x \mapsto \$g\}} && @(g(y),z) \equiv 1 \\
&\leadsto && @(y,z) \equiv 1 \\
&\leadsto_{\{y \mapsto \$g\}} && g(z) \equiv 1 \\
&\leadsto && z \equiv 1 \\
&\leadsto_{\{z \mapsto 1\}} && \mathit{true}.
\end{aligned}$$

Für die partiell applizierten Funktionen ergeben sich die Interpretationen $\$f \triangleq \lambda x'y'.f(x',y')$, $\$f(a) \triangleq \lambda x'.f(a,x')$ *und* $\$g \triangleq \lambda x'.g(x')$.
Zur Überprüfung der Korrektheit der Lösungen ergibt sich für die erste Substitution

$$\begin{aligned}
\big((\lambda x'y'.f(x',y'))a\big)b \equiv 1 &\to_\beta && (\lambda y'.f(a,y'))b \equiv 1 \\
&\to_\beta && f(a,b) \equiv 1 \\
&\to && 1 \equiv 1 \\
&\to && \mathit{true},
\end{aligned}$$

für die zweite

$$\begin{aligned}
\big((\lambda x'.g(x'))\lambda y'.f(a,y')\big)b \equiv 1 &\to_\beta && (g(\lambda y'.f(a,y'))b \equiv 1 \\
&\to && (\lambda y'.f(a,y'))b \equiv 1 \\
&\to_\beta && f(a,b) \equiv 1 \\
&\to && 1 \equiv 1 \\
&\to && \mathit{true},
\end{aligned}$$

und für die dritte

$$\begin{aligned}
\big((\lambda x'.g(x'))\lambda y'.g(y')\big)1 \equiv 1 &\to_\beta && (g(\lambda y'.g(y'))1 \equiv 1 \\
&\to && (\lambda y'.g(y'))1 \equiv 1 \\
&\to_\beta && g(1) \\
&\to && 1 \equiv 1 \\
&\to && \mathit{true}.
\end{aligned}$$

■

Die Ziele höherer Ordnung wurden mit expliziten Applikationssymbolen formuliert, ohne daß das Symbol @ Element der Signatur des (ursprünglichen) TES gewesen wäre. Bei der Übersetzung von TES $\mathcal{R}$ nach $\mathcal{R}^{@}$ wurde implizit davon ausgegangen, daß auf rechten Regelseiten keine nichtvariablen Terme mit Variablen als Kopfsymbol vorhanden waren, was beispielsweise die Implementierung der `map`-Funktion unmöglich machte.

Es ist nun naheliegend, in TES, die noch nicht mittels der Warren'schen Methode übersetzt wurden, ebenso wie in Zielen das explizite Applikationssymbol *mit einer Variablen als erstem Argument* auf rechten Regelseiten zuzulassen.

Beispiel 4.1.6.
Die Definitionen der `map`*-Funktion*

```
map _ [] = []
map F [X|Xs] = [F X|map F Xs] ,
```

die dem TES $\{map(F,[]) \to [], map(F,[X|Xs]) \to [@(F,X)|map(F,Xs)]$ *mit @ auf der zweiten rechten Regelseite entsprechen,*
wird in das TES

$$\left\{ \begin{array}{rcl} map(F,[]) & \to & [], \\ map(F,[X|Xs]) & \to & [@(F,X)|map(F,Xs)], \\ @(\$map,F) & \to & \$map(F), \\ @(\$map(F),L) & \to & map(F,L) \\ @(\$.,X) & \to & \$.(X) \\ @(\$.(X),Y) & \to & [X|Y] \end{array} \right\}$$

übersetzt, wobei die letzten zwei Regeln für den Listenkonstruktor '.' eingeführt wurden.
Fügt man zu diesen Regeln die in Bsp. 4.1.3 definierten Regeln für plus, inc, s und $\equiv$ *hinzu, so ergibt die Anfrage* $map(F,[0]) \equiv [s(0)]$ *die von NN errechneten Lösungen* $\{F \mapsto \$s\}$, $\{F \mapsto \$inc\}$ *und* $\{F \mapsto \$plus(s(0))\}$.

■

Das Beispiel läßt auch erahnen, daß es nicht notwendig ist, vollständig applizierte Funktionen (im Beispiel $map(F,[])$) mit expliziten Applikationen zu übersetzen, in diesem Fall also $@(@(\$map,F),[])$ (ohne Beweis; vgl. [AT98], die dieses Vorgehen als Optimierung vorschlagen).
Dieses Vorgehen ist in gewisser Weise natürlich unsauber: In einem TES wird eine Funktion (@) verwendet, die nicht definiert ist. Da die Lösungen aber immer bzgl. $\mathcal{R}^{@}$ berechnet werden, das Definitionen für @ enthält, ist es dennoch eine gangbare Lösung (es sei erneut darauf hingewiesen, daß sonst die Implementierung der *map*-Funktion auch nicht ohne weiteres möglich ist; vgl. den Abschnitt über applikative TES). Später werden auf rechten Regelseiten natürlich auch partiell applizierte Funktionen, d.h. die neu eingeführten Konstruktoren, zugelassen.

Weiter oben wurde erwähnt, daß nullstellige Symbole nicht codiert werden.

Beispiel 4.1.7.
Es sei $\{f \to 1\}$ *ein TES, das um die Regeln für die strikte Gleichheit erweitert wird. Lösungen für die Anfrage* $x \equiv 1$ *sind* $\{x \mapsto 1\}$ *und* $\{x \mapsto f\}$*. Die zweite Lösung kann unter Zuhilfenahme des vorgestellten Verfahrens nicht gefunden werden, weil die Funktion* f *nicht mit der Warren'schen Methode codiert werden kann. Eine Möglichkeit wäre, ein Dummy-Argument für die Funktion* f *einzuführen; da aber die Anfrage außer an Toplevel keine Narrowingpositionen besitzt, wird die ggf. gewünschte Lösung* $\{x \mapsto f\}$ *auch in diesem Fall nicht gefunden. Problematisch bei der Verwendung solcher Dummy-Argumente ist außerdem die Tatsache, daß bei der Auswertung Argumente von Termen unerlaubterweise eliminiert würden, was zu inkorrekten Lösungen (s.u.) führen kann.*

■

Die geschilderte Problematik ist nur bei nullstelligen Funktionssymbolen von Belang, weil Konstruktoren höherer Ordnung im nullstelligen Fall keinen Sinn machen. Es sei erneut darauf hingewiesen, daß nullstellige Symbole nicht partiell appliziert werden können.
Im Beispiel wurde von "inkorrekten Lösungen" gesprochen. Der Begriff der "Korrektheit" und "Vollständigkeit" ist in diesem Zusammenhang schwierig zu fassen: Bezüglich wessen ist eine Lösung korrekt oder vollständig? Und zeigt Beispiel 4.1.7, daß Warren's Methode unvollständig ist?
Die letzte Frage soll zuerst beantwortet werden: Dieses Beispiel ist kein Indiz für die "Unvollständigkeit", weil für funktional-logische Programmiersprachen geeignete Narrowingverfahren üblicherweise mit normalisierten Substitutionen umgegangen wird. Im Beispiel ist f nicht in Normalform.
Zur ersten Frage: Beide Begriffe sind natürlich sinnvollerweise bzgl. der betrachteten Ersetzungssysteme zu verstehen, da andernfalls in *jedem* Fall unendlich viele Lösungen gefunden werden müßten – man denke nur an die unendlich vielen Möglichkeiten, die Identitätsfunktion zu beschreiben; vgl. [ND94]. Die gleiche Problematik besteht natürlich auch im Narrowing erster Ordnung oder in der Resolution, wo als Trägermenge der Interpretation stets das (durch das Programm definierte) Herbranduniversum gewählt wird.
Damit ist das vorgestellte Verfahren natürlich in gewisser Weise "unvollständig": Bei Zugrundelegung der Regeln für die Addition aus 4.1.3 und für die *map*-Funktion aus Beispiel 4.1.6 wird für das Ziel $map(F, [0]) \equiv [0]$ zwar die Lösung $\{F \mapsto \$plus(0)\}$ mit der Interpretation $\lambda x.plus\ 0\ x$ gefunden, aber nicht die (symmetrische) Lösung $\lambda x.plus\ x\ 0$.
Damit stellt sich auch die Frage nach einer Korrektheit der Übersetzung: Inwiefern sind die TES $\mathcal{R}$ und $\mathcal{R}^{@}$ "äquivalent"? In [Nol95] wird die Äquivalenz beider Systeme im Kontext der funktionalen Programmierung bzgl. einer denotationellen Semantik (Abbildung in eine Algebra) bei beliebiger, aber fester

Reduktionsstrategie aufgezeigt. In [Gon93] wird die Korrektheit im Kontext der funktional-logischen Programmierung bzgl. der deklarativen Semantik von SFL [GHR92] in dem folgenden Sinn gezeigt: Jede korrekte und vollständige Narrowing-Strategie für SFL-Programme erster Ordnung (unter Verwendung von \$ und @) kann bei Elimination von @ und den \$-Konstruktoren in den Programmen in eine korrekte und vollständige Narrowing-Strategie für SFL-Programme höherer Ordnung (ohne @ und \$) übersetzt werden.
[AT98] definieren eine abstrakte funktional-logische Sprache, in der Konstrukte höherer Ordnung (Quellproblem) mit Warren's Methode (in das Zielproblem) übersetzt werden. Die Korrektheit ergibt sich dort als Kommutativität einer (diagrammatischen) Beziehung zwischen Quellproblem, Quellösung, Zielproblem und Ziellösung.

Daraus ergibt sich für die vorliegende Arbeit insbesondere die folgende Konsequenz:
Eine Zielsetzung dieser Arbeit ist der Vergleich der Kombination von NN und Warren's Methode mit einem Narrowing-Kalkül höherer Ordnung, NCA. Da die Korrektheit bzw. Vollständigkeit der Warren'schen Übersetzung schwierig zu fassen ist, ist die Simulation des zweiten Kalküls durch den ersten nicht ohne weiteres durch eine Simulation des ersten Verfahrens durch das zweite und bestimmte Eindeutigkeitseigenschaften nachzuweisen, wie das im Fall der Simulation von NN durch OINC möglich war.
Es wird später gelingen, einer NN-Ableitung eine NCA-Ableitung zuzuordnen. Diese Zuordnung kann als ein Korrektheitsbeweis von Warren's Methode angesehen werden, da NCA korrekt und vollständig ist.

Ein in dieser Arbeit nicht weiter verfolgter Ansatz ist die Übersetzung von NN in ein Schema von case-Ausdrücken, wie das in [HP96] durchgeführt wird. Dadurch wird NN in ein Inferenzschema transformiert, das ggf. leichter mit NCA zu vergleichen ist. Gelingt die wechselseitige Simulation auf diese Weise, so ergeben sich Korrektheit und Vollständigkeit der Warren'schen Transformation auf eine Weise, die der in [AT98] beschriebenen sehr nahe kommt.
Ein weiterer Ansatz ist die Formalisierung einer denotationellen Semantik für OINC bzw. NCA, die mit der in [HL96] vorgestellten denotationellen Semantik für NN verglichen werden könnte. Damit wäre die Zuordnung der Ableitungen zueinander allerdings nicht mehr konstruktiv.
Im Rahmen dieser Arbeit wird später versucht, die Simulation von NCA durch NN direkt zu zeigen, wie das für die Simulation von NN durch OINC und von NN durch NCA vorgeführt wurde.

Die Motivation für die eben erfolgten Ausführungen lag darin, daß anhand nullstelliger Funktionen Schwierigkeiten bei der Definition der Begriffe "Vollständigkeit" und "Korrektheit" zutage traten. Im bereits angesprochenen Kalkül NCA wird dieses Problem so gelöst, daß – wie üblich – NCA als vollständig in bezug auf normalisierbare Substitutionen definiert wird. Ein ähnliches Vorgehen bietet sich auch für die Kombination aus NN und Warren's Methode an. An dieser Stelle sei nur festgehalten, daß die Übersetzung für nullstellige Funktionen oder

Konstruktoren gewissermaßen unnötig ist.

4.2 Applikative TES

Die Verwendung von Warren's Methode wurde im letzten Abschnitt u.a. dadurch motiviert, daß die Notwendigkeit einer geeigneten Repräsentation von nichtvariablen Termen, die eine Variable als Kopfsymbol besitzen, erkannt wurde. Diese Repräsentation fußt auf der Einführung eines expliziten Applikationssymbols. Ein alternativer Ansatz, der Grundlage für den Kalkül NCA ist, besteht darin, die implizite Applikation von (Funktions-)Variablen implizit zu belassen. Daraus ergibt sich eine neue Definition von Termen, die Grundlage für die sog. applikativen TES ist.
Applikative TES bestehen aus Regeln mit *applikativen Termen.*

Definition 4.2.1. *Applikative Terme sind induktiv wie folgt definiert:*

- x *mit* $x \in \mathcal{V}ar$ *ist ein applikativer Term.*
- *Eine Konstante* c *ist ein applikativer Term.*
- *Ist* a *ein applikativer Term, so ist auch* (a) *ein applikativer Term.*
- *Wenn* a_1 *und* a_2 *applikative Terme sind, dann ist auch* $a_1 a_2$ *ein applikativer Term.*
- $\Box$ *ist ein applikativer Term (der allerdings nur "alleine" auftritt).*

$\triangle$

Dem Nebeneinanderstellen zweier applikativer Terme liegt die Vorstellung zugrunde, daß diese Operation die (implizite) Applikation des ersten auf den zweiten Term darstellt. Dieses versteckte Applikationssymbol ist linksassoziativ und wird stets als infix notiert angenommen. Auffällig ist die Tatsache, daß nicht zwischen Funktions- und Konstruktorsymbolen unterschieden wird. Diese Unterscheidung wird auch in [IN94] bei der Definition von OINC nicht getroffen, wurde aber – zum Zweck der Vergleichbarkeit mit NN – im Rahmen dieser Arbeit eingeführt. Aus demselben Grund wird auch hier zwischen Funktions- und Konstruktorsymbolen unterschieden, deren Vereinigung die Menge der Konstanten ergibt.

Beispiel 4.2.2. *(applikative Terme)*
Die Ausdrücke x, $c\ x$, $f\ (c\ a)\ b$ *und* $cons\ (F\ x)\ (map\ F\ xs)$ *sind applikative Terme, wobei* $\{F, x, xs\} \subseteq \mathcal{V}ar$ *ist und die restlichen Symbole Konstanten sind.*

Drückt man die implizite Applikation durch ein explizites Applikationssymbol @ aus, so sind diese Terme gleichbedeutend mit x, $@(c, x)$, $@(@(f, @(c, a)), b)$ und $@(@(cons, @(F, x)), @(@(map, F), xs))$.

■

Die leftmost-innermost-Position eines applikativen Terms ist die leftmost-innermost-Position des Terms mit expliziter Notation der Applikation.
Mit applikativen Termen können applikative TES erzeugt werden. Dazu benötigt man zunächst den Begriff des Musters:

Definition 4.2.3. *(Muster in applikativen Systemen, [NMI95])*
Ein Muster ist ein applikativer Term t mit der Eigenschaft, daß das leftmost-innermost Symbol jedes nichtvariablen Unterterms von t eine Konstante ist.

△

Damit ist ein Muster entweder eine Variable oder ein Term der Form $f\ t_1 \dots t_n$, wobei $f \notin \mathcal{V}ar$ ist und $t_1, \dots, t_n$ Muster sind.

Beispiel 4.2.4. *(Muster, [NMI95])*
Der Term $+\ (s\ 0)\ (+\ x)$ ist ein Muster, aber der Term $s\ (x\ 0)$ mit $x \in \mathcal{V}ar$ ist keins.

■

Definition 4.2.5. *(Applikative Ersetzungsregel, Applikatives TES, [NMI95])*

- *Eine applikative Ersetzungsregel ist ein Paar $l \rightarrow r$ applikativer Terme, so daß die linke Regelseite l ein Muster der Form $f\ l_1 \dots l_n$ mit $n = Arity(f)$ und $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ ist.*
- *Ein applikatives TES (ATES) besteht aus einer (endlichen) Menge von applikativen Ersetzungsregeln.*

△

Begriffe wie "Stelligkeit", "Orthogonalität", "Konstruktorbasiertheit" usf. lassen sich intuitiv auf ATES anwenden.

Beispiel 4.2.6. *(ATES, vgl. [NMI95])*
Die Menge $\{map\ F\ nil \rightarrow nil, map\ F\ (cons\ x\ xs) \rightarrow cons\ (F\ x)(map\ F\ xs)\}$ ist ein ATES mit $Arity(map) = 2$. Die Argumente der linken Seiten enthalten das (implizite) Applikationssymbol. So sind die Argumente der ersten linken Regelseite $map\ F$ und nil, nicht F und nil.

■

Im folgenden wird wieder die übliche Notation für Listen mit eckigen Klammern verwendet. Eine Liste besteht dann aus dem Resultat der Applikation des Listenkonstruktors auf den Listenkopf, das auf den Listenrumpf appliziert wird.

Außerdem werden applikative Terme $f\ t_1 \dots t_n$ ggf. mit $f\overline{t_n}$ abgekürzt.

4.3 Eine Erweiterung von OINC: NCA

OINC ist korrekt und vollständig für beliebige orthogonale TES. Damit kann OINC insbesondere für das Rechnen mit orthogonalen applikativen TES verwendet werden.

Beispiel 4.3.1. *(OINC mit ATES)*
Es sei $\mathcal{R} = \{plus\ 0\ x \to x, plus\ (s\ x)y \to s\ (plus\ x\ y)\}$ *ein (orthogonales) ATES.*
Dann gibt es für die Anfrage $x\ 0\ y \approx s\ 0$ *die OINC-Ableitung*

$$\begin{array}{lll} x\ 0\ y \approx s\ 0 & \rightsquigarrow_{[on]} & x\ 0 \approx plus\ 0, y \approx x', x' \approx s\ 0 \\ & \rightsquigarrow_{[d]} & x \approx plus, 0 \approx 0, y \approx x', x' \approx s\ 0 \\ & \rightsquigarrow_{[v],\{x \mapsto plus\}} & 0 \approx 0, y \approx x', x' \approx s\ 0 \\ & \rightsquigarrow_{[d]} & y \approx x', x' \approx s\ 0 \\ & \rightsquigarrow_{[v],\{x' \mapsto y\}} & y \approx s\ 0 \\ & \rightsquigarrow_{[v],\{y \mapsto s\ 0\}} & \square \end{array}$$

mit der Lösung $\{x \mapsto plus, y \mapsto s\ 0\}$.

■

Wiederum können die partiellen Funktionen als λ-Ausdrücke interpretiert werden. Im Beispiel entspricht die Bindung von x an *plus* der nicht applizierten Funktion $\lambda x'y'.plus\ x'\ y'$.

In [NMI95] wird gezeigt, daß OINC applikative Terme sehr ineffizient behandelt, weil das Toplevel-Symbol der linken Gleichungsseite fast immer das (implizite) Applikationssymbol ist, das eigentlich keine Information trägt. Daraus resultiert die Definition eines neuen Kalküls:

Definition 4.3.2. *(NCA, [NMI95])*
Der Kalkül NCA (Narrowing Calculus for Applicative Systems) ist durch folgendes Inferenzschema definiert:

- *[ona], outermost Narrowing applikativer Terme*

$$\frac{f\overline{s_n}\overline{t_m} \approx t, E}{s_1 \approx u_1, \dots, s_n \approx u_n, r\overline{t_m} \approx t, E}, t \notin \mathcal{V}ar$$

für eine frische Regel $f\overline{u_n} \to r$*;*

- *[onv], outermost Narrowing für Terme mit Variablen als Kopfsymbol*

$$\frac{x\overline{s_n}\,\overline{t_m} \approx t, E}{\vartheta(s_1 \approx v_1, \ldots, s_n \approx v_n, r\overline{t_m} \approx t, E)}, t \notin \mathcal{V}ar$$

für eine frische Regel $f\overline{u_k v_n} \to r$, $n > 0$ *und* $\vartheta = \{x \mapsto f\overline{u_k}\}$*;*

- *[da], Dekomposition applikativer Terme*

$$\frac{f\overline{s_n} \approx f\overline{t_n}, E}{s_1 \approx t_1, \ldots, s_n \approx t_n, E};$$

- *[dv], Dekomposition von Termen mit Variablen als Kopfsymbol*

$$\frac{x\overline{s_n} \approx f\overline{t_m}\,\overline{u_n}, E}{\vartheta(s_1 \approx u_1, \ldots, s_n \approx u_n, E)}$$

mit $\vartheta = \{x \mapsto f\overline{t_m}\}$*;*

- *[v], Variablenelimination*

$$\frac{t \approx x, E}{\vartheta(E)}$$

mit $\vartheta = \{x \mapsto t\}$.

$\triangle$

Beispiel 4.3.3. *(Fortsetzung von Bsp. 4.3.1)*
Es sei $\mathcal{R}$ *das ATES aus Beispiel 4.3.1. Für die Anfrage* $x\ 0\ y \approx s\ 0$ *ergibt sich die NCA-Ableitung*

$$\begin{array}{lll} x\ 0\ y \approx s\ 0 & \rightsquigarrow_{[onv],\{x \mapsto plus\}} & 0 \approx 0, y \approx x', x' \approx s\ 0 \\ & \rightsquigarrow_{[da]} & y \approx x', x' \approx s\ 0 \\ & \rightsquigarrow_{[v],\{x' \mapsto y\}} & y \approx s\ 0 \\ & \rightsquigarrow_{[dv],\{y \mapsto s\ 0\}} & \square \end{array}$$

$\blacksquare$

Beispiel 4.3.4. *(Fortsetzung von Bsp. 4.1.5)*
Es sei $\mathcal{R}'$ *das TES aus Beispiel 4.1.5 in "applikativer Schreibweise", d.h. die Terme werden als applikative Terme aufgefaßt.*
Dann gibt es bzgl. $\mathcal{R}'_{\equiv}$ *die NCA-Ableitungen*

$$\begin{array}{lll}
x\ y\ z \equiv 1 \approx true & \leadsto_{[ona]} & x\ y\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[onv],\{x \mapsto f\}} & y \approx a, z \approx b, 1 \approx 1, true \approx true \\
& \leadsto^{4}_{\{y \mapsto a, z \mapsto b\}} & \Box,
\end{array}$$

$$\begin{array}{lll}
x\ y\ z \equiv 1 \approx true & \leadsto_{[ona]} & x\ y\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[onv],\{x \mapsto g\}} & y \approx x', x'\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[v],\{x' \mapsto y\}} & y\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[onv],\{y \mapsto f\ a\}} & z \approx b, 1 \approx 1, 1 \approx 1, true \approx true \\
& \leadsto^{4}_{\{z \mapsto b\}} & \Box \quad \textit{und}
\end{array}$$

$$\begin{array}{lll}
x\ y\ z \equiv 1 \approx true & \leadsto_{[ona]} & x\ y\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[onv],\{x \mapsto g\}} & y \approx x', x'\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[v],\{x' \mapsto y\}} & y\ z \approx 1, 1 \approx 1, true \approx true \\
& \leadsto_{[onv],\{y \mapsto g\}} & z \approx x'', x'' \approx 1, 1 \approx 1, true \approx true \\
& \leadsto^{4}_{\{x'' \mapsto z, z \mapsto 1\}} & \Box.
\end{array}$$

NN (und damit OINC) benötigt für die Berechnung der ersten Lösung bzgl. $\mathcal{R}^{@}_{\equiv}$ vier und für die Berechnung der zweiten und dritten Lösung jeweils fünf Narrowingschritte.

■

In [NMI95] wird nachgewiesen, daß eine NCA-Ableitung immer kürzer als eine entsprechende OINC-Ableitung mit derselben Lösung ist.
NCA ist vollständig und korrekt:

Satz 4.3.5. *(Vollständigkeit von NCA, Theorem 23 in [NMI95])*
$\mathcal{R}$ sei ein orthogonales ATES und G eine rechtsnormale Anfrage. Für jede normalisierbare Lösung ϑ von G gibt es eine erfolgreiche NCA-Ableitung $G \leadsto^*_{\vartheta'} \Box$, so daß $\vartheta' \leq_{\mathcal{R}} \vartheta[\mathcal{V}ar(G)]$ ist.

Satz 4.3.6. *(Korrektheit von NCA, Theorem 9 in [NMI95])*
$\mathcal{R}$ sei ein orthogonales ATES und G eine rechtsnormale Anfrage. Wenn es eine erfolgreiche NCA-Ableitung $G \leadsto^*_{\vartheta} \Box$ gibt, dann ist ϑ eine Lösung für G.

Wie für OINC wird der Begriff der NCA-geeigneten Gleichungsmenge definiert:

Definition 4.3.7. *(NCA-geeignet)*
Eine Gleichungsmenge G heißt NCA-geeignet, wenn es eine rechtsnormale Gleichungsmenge G' und eine NCA-Ableitung $G' \leadsto^* G$ gibt.

△

4.4 Simulation von NN durch NCA

Um beide Kalküle miteinander vergleichen zu können, müssen zunächst die betrachteten TES $\mathcal{R}$ und $\mathcal{R}^@$ zueinander in Beziehung gesetzt werden. Dies geschieht mit der bijektiven Transformationsfunktion τ. Im folgenden werden der Übersichtlichkeit halber @-Symbole indiziert, um ihre Anzahl anzugeben (es ist also $@_n(\cdots @_1(s_0, s_1) \cdots, s_n)$ für $n = 2$ der Term $@(@(s_0, s_1), s_2)$).

Definition 4.4.1. *(Transformationen τ, τ^{-1})*
Die Transformationsfunktion τ ist eine Abbildung von der Menge der applikativen Terme über $\Sigma \cup \{\Box, @\}$ mit $\Sigma = \mathcal{F} \cup \mathcal{C}$ in die Menge der Terme über $\Sigma \cup \Sigma^\$ \cup \{@, \Box\}$, wobei $\Sigma^\$$ aus Σ ensteht, indem für jedes Symbol f aus Σ der Stelligkeit n n neue (polymorphe) Konstruktoren $\$f$ der Stelligkeiten $0, 1, 2, \ldots, n-1$ eingeführt werden. Dabei werde in den applikativen Termen zwischen Funktionssymbolen $\mathcal{F}$ und Konstruktorsymbolen $\mathcal{C}$ unterschieden.
Im einzelnen ist τ wie folgt induktiv definiert:

$$\begin{aligned}
\tau(\Box) &= \Box, \\
\tau(x) &= x, \text{ falls } x \in \mathcal{V}ar, \\
\tau(c) &= c, \text{ falls } c \in \Sigma \text{ die Stelligkeit 0 hat}, \\
\tau(f\overline{s_n}) &= \$f(\tau(s_1), \ldots, \tau(s_n)), \text{ falls } f \in \Sigma \text{ mit } 1 \le n < Arity(f) \text{ ist}, \\
\tau(f\overline{s_n}) &= f(\tau(s_1), \ldots, \tau(s_n)), \text{ falls } f \in \Sigma \text{ mit } 1 \le n = Arity(f) \text{ ist}, \\
\tau(f\overline{s_n}\overline{t_m}) &= @_m\big(@_{m-1}\big(\cdots @_2(@_1(\tau(f\overline{s_n}), \tau(t_1)), \tau(t_2)) \cdots, \tau(t_{m-1})\big), \tau(t_m)\big), \\
&\quad\ \text{falls } f \in \Sigma, n = Arity(f) > 0 \text{ und } m > 0 \text{ ist}, \\
\tau(x\overline{s_n}) &= @_n\big(\cdots @_2\big(@_1(x, \tau(s_1)), \tau(s_2)\big) \cdots, \tau(s_n)\big), \\
&\quad\ \text{falls } x \in \mathcal{V}ar \textit{ und } n > 0 \textit{ ist.}
\end{aligned}$$

Die inverse Transformation τ^{-1} ergibt sich direkt aus der Definition von τ.

△

Beispiel 4.4.2. *(τ, τ^{-1})*
Es sei $Arity(f) = Arity(g) = 2$ und $x \in \mathcal{V}ar$. Dann ist $\tau(f\ x\ (g\ a\ b)) = f(x, g(a, b)), \tau(f\ a) = \$f(a), \tau(x, a) = @(x, a)$ und $\tau(f\ a\ b\ c) = @(f(a, b), c)$ bzw. $\tau^{-1}(f(x, g(a, b))) = f\ x\ (g\ a\ b), \tau^{-1}(\$f(a)) = f\ a, \tau^{-1}(@(x, a) = x\ a$ und $\tau^{-1}(@(f(a, b), c)) = f\ a\ b\ c$.

■

Da NN für induktiv-sequentielle TES definiert ist, können nur solche TES miteinander verglichen werden, bei denen das mit der Warren'schen Methode transformierte TES ein induktiv-sequentielles System liefert.

Definition 4.4.3. *(flache applikative Systeme)*
Ein ATES $\mathcal{R}$ heißt flaches applikatives TES, falls $\mathcal{S}^@$, das entsprechende System in nicht-applikativer Schreibweise erweitert um die durch die Warren'sche Transformation neu entstandenen Regeln, induktiv-sequentiell ist. "Nicht-applikative" Schreibweise bedeutet dabei, daß die Terme nicht als applikative

Terme, sondern als Terme im üblichen Sinn aufgefaßt werden (die Applikation @ auf rechten Regelseiten ist gestattet).

$\triangle$

Mit dieser (intuitiv eingängigen) Transformation der Ersetzungssysteme können nun die einzelnen Inferenzen von NCA bzgl. $\mathcal{R}$ mit Inferenzen von NN oder OINC bzgl. $\mathcal{R}^@$ verglichen werden.

Da die von OINC berechneten Ableitungen nach [NMI95] in jedem Fall länger als die von NCA berechneten sind und außerdem das (zeitliche) Rechenverhalten für ein angegebenes Beispiel um den Faktor 30 schlechter ist (bei ungefährer Verdopplung der Schrittzahlen), wird im folgenden nur NCA bzw. s-NCA untersucht.
Der Simulationsbeweis gestaltet sich schwieriger als die Simulation von OINC und NN, weil jetzt nicht nur verschiedene Kalküle in Beziehung gesetzt werden, sondern außerdem unterschiedliche Termstrukturen (applikativ vs. "herkömmlich") und unterschiedliche zugrundeliegende TES ($\mathcal{R}$ vs. $\mathcal{R}^@$) beachtet werden müssen. Für den ersten Aspekt ergibt sich die Frage, wie man Positionen in applikativen und "herkömmlichen" Termen vergleichen kann.

Beispiel 4.4.4. *(unterschiedliche Termstrukturen und Positionen)*
Der applikative Term $t = f\ x\ (g\ a\ b)$ *besitzt mit expliziter Notation der Applikation* ($t^@ = @\big(@(f,x), @(@(g,a),b)\big)$) *die Unterterme* $t^@|_1 = @(f,x)$, $t^@|_{1\cdot 1} = f, t^@|_{1\cdot 2} = x, t^@|_2 = @(@(g,a),b), t^@|_{2\cdot 1} = @(g,a), t^@|_{2\cdot 2} = b, t^@|_{2\cdot 1\cdot 1} = g, t^@|_{2\cdot 1\cdot 2} = a$ *und damit die Positionen* $\mathcal{O} = \{\varepsilon, 1, 2, 1\cdot 1, 1\cdot 2, 2\cdot 1, 2\cdot 2, 2\cdot 1\cdot 1, 2\cdot 1\cdot 2\}$.
Der entsprechende Term $f(x, g(a,b))$ *besitzt die Positionsmenge* $\mathcal{O}' = \{\varepsilon, 1, 2, 2\cdot 1, 2\cdot 2\}$.
Mit Hilfe der Transformationsfunktion τ *können die Positionen verglichen werden:* $\tau(f\ x\ (g\ a\ b)) = f(x, g(a,b))$ *liefert sofort* $\mathcal{O}(\tau(f\ x\ (g\ a\ b)) = \mathcal{O}'$.

$\blacksquare$

Wenn im folgenden Unterterme applikativer Terme adressiert werden, dann ist darunter stets die Position des mit τ transformierten Terms zu verstehen, d.h. für einen applikativen Term t ist $t|_p = \tau^{-1}(\tau(t)|_p)$.

Beispiel 4.4.5. *(Adressierung applikativer Terme)*
$t = f\ x\ (g\ a\ b)$ *mit* $\tau(t) = f(x, g(a,b))$ *sei ein applikativer Term. Dann ist* $t|_1 = \tau^{-1}(\tau(t)|_1) = x$ *und nicht* $f\ x$, $t|_{2\cdot 1} = \tau^{-1}(\tau(t)|_{2\cdot 1}) = a$ *und nicht* $g\ a\ b$.
Für ein einstelliges g *sei* $s = f\ (g\ a\ b)$ *mit* $\tau(s) = f(@(g(a), b)$. *Dann ist* $s|_1 = \tau^{-1}(\tau(s)|_1) = g\ a\ b$ *und nicht* f *und* $s|_{1\cdot 1} = \tau^{-1}(\tau(s)|_{1\cdot 1}) = g\ a$, *wobei diese Position in* s *nicht existiert.*

$\blacksquare$

Im folgenden bezeichnet $\overrightarrow{a_n}$ das Tupel $a_1, \ldots, a_n$.
Wenn ein NN-Schritt mit einer Regel $@(\$c(\overrightarrow{x_k}), x_{k+1}) \rightarrow [\$]c(\overrightarrow{x_{k+1}})$ (mit dem optionalen Dollarzeichen, falls $k+1 = m := Arity(c)$ ist) auf einen Term $@_n(\cdots @_1(t_1, t_2), \cdots, t_n)$ angewendet wird, dann sind damit die nächsten $min(n, m-k-1)$ Schritte deterministisch festgelegt (auf rechten Regelseiten stehen nur partiell applizierte Konstruktoren). Die Idee für den Simulationsbeweis liegt nun darin, diese (deterministische) Folge von Schritten zu einem "Multischritt" zusammenzufassen.

Definition 4.4.6. *(Multischritt, NN-Multiableitung, $\looparrowright$)*

1. *Eine maximale Folge von NN-Schritten*

$$t_1 \rightsquigarrow_{p_1, l_1 \rightarrow r_1, \sigma_1} t_2 \rightsquigarrow \cdots t_n \rightsquigarrow_{p_n, l_n \rightarrow r_n, \sigma_n} t_{n+1}$$

   heißt

   - *f-Multischritt der Länge n, falls*
     (a) $l_i \rightarrow r_i = @(\$f(\overrightarrow{x_{k_i}}), x_{k_i+1}) \rightarrow r_i$ mit $f \in \mathcal{F}$ für $1 \leq i < n$ und
     (b) $p_{i+1} = p_i \cdot 1$ für $1 \leq i < n-1$ und $p_n = p_{n-1}$ ist und
   - *c-Multischritt der Länge n, falls*
     (a) $l_i \rightarrow r_i = @(\$c(\overrightarrow{x_{k_i}}), x_{k_i+1}) \rightarrow r_i$ mit $c \in \mathcal{C}$ für $1 \leq i < n$ und
     (b) $p_{i+1} = p_i \cdot 1$ für $1 \leq i < n$ ist.

2. *Ein Narrowingschritt $\rightsquigarrow_{p,l \rightarrow r,\sigma}$ mit $Head(l) \neq @$ heißt 0-Multischritt, falls im Schritt vorher keine Regel für $@$ angewendet wurde (notiert als $\looparrowright^0_{p,l \rightarrow r,\sigma}$).*

3. *Ein C-Multischritt ist eine maximale Folge von c-Multischritten.*

4. *Ein F-Multischritt ist ein evtl. leerer C-Multischritt gefolgt von einem f-Multischritt (notiert als $\looparrowright^F_{p,l \rightarrow r,\sigma}$, wobei p und $l \rightarrow r$ Position und Regel des f-Multischritts sind und σ die Komposition der Substitutionen der einzelnen Schritte ist).*

5. *Ein F0-Multischritt ist ein evtl. leerer C-Multischritt gefolgt von einem 0-Multischritt (notiert als $\looparrowright^{F0}_{p,l \rightarrow r,\sigma}$, wobei p und $l \rightarrow r$ Position und Regel des 0-Multischritts sind und σ die Komposition der Substitutionen der einzelnen Schritte ist).*

6. *Ein Multischritt ist ein 0-, C-, F- oder F0-Multischritt und wird mit $\looparrowright$ gekennzeichnet (evtl. indiziert mit Position, Regel oder Substitution).*

7. *Eine Folge von Multischritten ist eine NN-Multiableitung.*

$\triangle$

F0-Multischritte entsprechen dem Binden von Variablen an partiell applizierte Konstruktoren und einem abschließenden "normalen" NN-Narrowingschritt:

Beispiel 4.4.7. *(F0-Multischritte)*
Bzgl. des TES $\mathcal{R} = \{f(c(a,b,d)) \to a\}$ *gibt es die Multiableitungen*

$$\begin{array}{lll} f(@(x,y)) \equiv a & \hookrightarrow^{F0}_{\varepsilon,\{x\mapsto\$c(a,b)\}} & a \equiv a \\ & \hookrightarrow^{0} & true, \end{array}$$

$$\begin{array}{lll} f(@(@(x,y),z)) \equiv a & \hookrightarrow^{F0}_{\varepsilon,\{x\mapsto\$c(a),y\mapsto b,z\mapsto d\}} & a \equiv a \\ & \hookrightarrow^{0} & true \;\; \text{und} \end{array}$$

$$\begin{array}{lll} f(@(@(x,y),z),z') \equiv a & \hookrightarrow^{F0}_{\varepsilon,\{x\mapsto\$c,y\mapsto a,z\mapsto b,z'\mapsto d\}} & a \equiv a \\ & \hookrightarrow^{0} & true \end{array}$$

Die entsprechenden NN-Ableitungen werden in Beispiel 4.5.5 angegeben.

■

F-Multischritte entsprechen einer Ableitung, in der zunächst ggf. (mehrere verschiedene) Variablen an (mehrere verschiedene) partiell applizierte Konstruktoren gebunden werden und danach eine Variable an eine partiell applizierte Funktion:

Beispiel 4.4.8. *(F-Multischritte)*
Den Ableitungen aus Beispiel 4.1.5 bzgl. des TES $\mathcal{R} = \{f(a,b) \to 1, g(x) \to x\}$ *entsprechen die Multiableitungen*

$$\begin{array}{lll} @(@(x,y),z) \equiv 1 & \hookrightarrow^{F}_{\{\varepsilon,f(a,b)\to 1,\{x\mapsto\$f,y\mapsto a,z\mapsto b\}} & 1 \equiv 1 \\ & \hookrightarrow^{0} & true, \end{array}$$

$$\begin{array}{lll} @(@(x,y),z) \equiv 1 & \hookrightarrow^{F}_{1,g(x)\to x,\{x\mapsto\$g\}} & @(y,z) \equiv 1 \\ & \hookrightarrow^{F}_{\varepsilon,\{y\mapsto\$f(a),z\mapsto b\}} & 1 \equiv 1 \\ & \hookrightarrow^{0} & true \;\; \text{und} \end{array}$$

$$\begin{array}{lll} @(@(x,y),z) \equiv 1 & \hookrightarrow^{F}_{1,g(x)\to x,\{x\mapsto\$g\}} & @(y,z) \equiv 1 \\ & \hookrightarrow^{F}_{\varepsilon,g(x)\to x,\{y\mapsto\$g\}} & z \equiv 1 \\ & \hookrightarrow^{0}_{\{z\mapsto 1\}} & true \end{array}$$

■

Entscheidend ist die Idee, *sämtliche* Anwendungen von Regeln für @ zu einem Multischritt zusammenfassen. Das ermöglicht nämlich später die Simulation durch [dv]-Inferenzen.

Beispiel 4.4.9. *(F- und c-Multischritte, Korrespondenz mit NCA)*
$\mathcal{R} = \{f\ (c\ a\ b)\ a \to a, g\ a \to a\}$ *sei ein ATES und*

$$\mathcal{S}^{@}_{\equiv} \supset \left\{ \begin{array}{l} f(c(a,b),a) \to a, \\ g(a) \to a, \\ @(\$g, x_1) \to g(x_1), \\ @(\$c, x_1) \to \$c(x_1), @(\$c(x_1), x_2) \to c(x_1, x_2) \end{array} \right\}$$

das entsprechende System in "herkömmlicher" Schreibweise, erweitert um die neuen Regeln. Dann gibt es die NN-Ableitung bzgl. $\mathcal{S}^{@}_{\equiv}$

$$\begin{array}{lll} f(@(x,y), @(z,a)) \equiv a & \rightsquigarrow_{\{x \mapsto \$c(x')\}} & f(c(x',y), @(z,a)) \equiv a \\ & \rightsquigarrow_{\{x' \mapsto a, y \mapsto b, z \mapsto \$g\}} & f(c(a,b), g(a)) \equiv a \\ & \rightsquigarrow & f(c(a,b),a) \equiv a \\ & \rightsquigarrow & a \equiv a \\ & \rightsquigarrow & true, \end{array}$$

wobei der erste Schritt einen c-, der zweite einen f- und die ersten drei zusammen einen F-Multischritt bilden:

$$\begin{array}{lll} f(@(x,y), @(z,a)) \equiv a & \looparrowright^{F}_{2, g(a) \to a, \{x \mapsto \$c(a), y \mapsto b, z \mapsto \$g\}} & f(c(a,b),a) \equiv a \\ & \looparrowright^{0} & a \equiv a \\ & \looparrowright^{0} & true. \end{array}$$

Außerdem gibt es die NCA-Ableitung bzgl. $\mathcal{R}_{\equiv}$

$$\begin{array}{lll} f\ (x\ y)\ (z\ a) \equiv a \approx true & \rightsquigarrow_{[ona]} & f\ (x\ y)\ (z\ a) \approx a, a \approx a, \\ & & true \approx true \\ & \rightsquigarrow_{[ona]} & x\ y \approx c\ a\ b, z\ a \approx a, a \approx a, \\ & & true \approx true \\ & \rightsquigarrow_{[dv], \{x \mapsto c\ a\}} & y \approx b, z\ a \approx a, a \approx a, true \approx true \\ & \rightsquigarrow_{[dv], \{y \mapsto b\}} & z\ a \approx a, a \approx a, true \approx true \\ & \rightsquigarrow_{[onv], \{z \mapsto g\}} & a \approx a, a \approx a, a \approx a, true \approx true \\ & \rightsquigarrow^{4} & \square \end{array}$$

Man mache sich insbesondere den Zusammenhang zwischen dem c-Multischritt und der ersten [dv]-Inferenz sowie zwischen dem F-Multischritt und den beiden [dv]- und der [onv]-Inferenz klar: Um den [onv]-Schritt durchführen zu können, muß zunächst die Gleichung $x\ y \approx c\ a\ b$ *gelöst werden, was dem c-Multischritt aus dem F-Multischritt entspricht. Die Bindung von* x *und* y *in der NN-Inferenz findet bei Traversieren des LR-DT für* f *statt.*

■

Es liegt nun nahe, daß die Ergebnisse der Simulation von NN durch OINC mit dieser neuen Notation auf NCA übertragbar sind.

Beispiel 4.4.10.
$\mathcal{R}$ sei das ATES für die Addition.
Bzgl. $\mathcal{R}_\equiv$ (strikte Gleichheit infix notiert) gibt es die NCA-Ableitung

$$\begin{array}{rll}
x\ 0\ y \equiv s\ 0 \approx true & \rightsquigarrow_{[ona1]} & x\ 0\ y \approx s\ x', s\ 0 \approx s\ y', x' \equiv y' \approx true \\
& \rightsquigarrow_{[onv2],\{x \mapsto plus\}} & 0 \approx 0, y \approx x'', x'' \approx s\ x', s\ 0 \approx s\ y', \\
& & x' \equiv y' \approx true \\
& \rightsquigarrow_{[da]} & y \approx x'', x'' \approx s\ x', s\ 0 \approx s\ y', \\
& & x' \equiv y' \approx true \\
\Longrightarrow^2 & \rightsquigarrow_{[v],\{x'' \mapsto y\}} & y \approx s\ x', s\ 0 \approx s\ y', x' \equiv y' \approx true \\
& \rightsquigarrow_{[v],\{y \mapsto s\ x'\}} & s\ 0 \approx s\ y', x' \equiv y' \approx true \\
& \rightsquigarrow_{[da]} & 0 \approx y', x' \equiv y' \approx true \\
\Longrightarrow^1 & \rightsquigarrow_{[v],\{y' \mapsto 0\}} & x' \equiv 0 \approx true \\
& \rightsquigarrow_{[ona3]} & x' \approx 0, true \approx true \\
\Longrightarrow^3 & \rightsquigarrow_{[dv],\{x' \mapsto 0\}} & true \approx true \\
& \rightsquigarrow_{[da]} & \square
\end{array}$$

Ohne den Begriff der [ona]- und [onv]-Lösungen zu formalisieren, ist bei einer intuitiven Auffassung ihrer Bedeutung die bei $\Longrightarrow^2$ inferierte Gleichungsmenge [onv]-gelöst bzgl. des einzigen [onv]-Schritts, und die bei $\Longrightarrow^1$ bzw. $\Longrightarrow^3$ inferierte Gleichungsmenge ist [ona]-gelöst bzgl. ersten bzw. zweiten [ona]-Schritts an der Wurzelposition.
Für das korrespondierende induktiv-sequentielle TES

$$\mathcal{S} = \{plus(0,x) \rightarrow x, plus(s(x),y) \rightarrow s(plus(x,y)\}$$

und die korrespondierende Anfrage $@(@(x,0),y) \equiv s(0)$ gibt es bzgl. $\mathcal{S}^@_\equiv$ die NN-Ableitung (die stellvertetend für die similare OINC-Ableitung angegeben wird)

$$\begin{array}{rll}
@(@(x,0),y) \equiv s(0) & \rightsquigarrow_{\{x \mapsto \$plus\}} & @(\$plus(0),y) \equiv s(0) \\
& \rightsquigarrow & plus(0,y) \equiv s(0) \\
& \rightsquigarrow & y \equiv s(0) \\
& \rightsquigarrow_{\{y \mapsto s(x')\}} & x' \equiv 0 \\
& \rightsquigarrow_{\{x' \mapsto 0\}} & true
\end{array}$$

mit der resultierenden Substitution $\{x \mapsto \$plus, y \mapsto s(0)\}$.
Dabei entsprechen die ersten drei Schritte der [onv]-Inferenz bzw. ihrer Lösung in der NCA-Ableitung, der vierte Schritt entspricht der ersten und der fünfte der zweiten [ona]-Inferenz bzw. deren Lösung.

Die korrespondierende Multiableitung ist von der Form

$$\begin{array}{rll}
@(@(x,0),y) \equiv s(0) & \looparrowright^F_{\varepsilon, plus(0,x) \rightarrow x, \{x \mapsto \$plus\}} & y \equiv s(0) \\
& \looparrowright^0_{\{y \mapsto s(x')\}} & x' \equiv 0 \\
& \looparrowright^0_{\{x' \mapsto 0\}} & true
\end{array}$$

■

Beispiel 4.4.11.
Für ein ATES mit Regeln für map und plus gibt es die NCA-Ableitung

$$
\begin{array}{lll}
\multicolumn{3}{l}{map\ F\ [0] \equiv [0] \approx true} \\
 & \leadsto_{[ona1]} & map\ F\ [0] \approx [x_1|y_1], [0] \approx [x_2|y_2], \\
 & & x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx true \\
 & \leadsto_{[ona2]} & F \approx F', [0] \approx [x'|y'], [F'\ x'|map\ F'\ y'] \approx [x_1|y_1], \\
 & & [0] \approx [x_2|y_2], x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx true \\
\Longrightarrow^2 & \leadsto^4_{\{F'\mapsto F, x'\mapsto 0, y'\mapsto []\}} & [F\ 0|map\ F\ []] \approx [x_1|y_1], [0] \approx [x_2|y_2], \\
 & & x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx true \\
 & \leadsto^3_{\{x_1\mapsto F0, y_1\mapsto map\ F[]\}} & [0] \approx [x_2|y_2], F\ 0 \equiv x_2 \wedge map\ F\ [] \equiv y_2 \approx true \\
\Longrightarrow^1 & \leadsto^3_{\{x_2\mapsto 0, y_2\mapsto []\}} & F\ 0 \equiv 0 \wedge map\ F\ [] \equiv [] \approx true \\
 & \leadsto_{[ona3]} & F\ 0 \equiv x_2 \approx true, map\ F\ [] \equiv y_2 \approx x'', x'' \approx true \\
 & \leadsto_{[ona4]} & F\ 0 \approx 0, x_2 \approx 0, true \approx true, \\
 & & map\ F\ [] \equiv y_2 \approx x'', x'' \approx true \\
 & \leadsto_{[onv5],\{F\mapsto plus 0\}} & 0 \approx 0, 0 \approx 0, x_2 \approx 0, true \approx true, \\
 & & map\ F\ [] \equiv y_2 \approx x'', x'' \approx true \\
\Longrightarrow^5 & \leadsto_{[da]} & 0 \approx 0, x_2 \approx 0, true \approx true, \\
 & & map\ F\ [] \equiv y_2 \approx x'', x'' \approx true \\
\Longrightarrow^4 & \leadsto^3_{\{x_2\mapsto 0\}} & true \approx true, map\ F\ [] \equiv y_2 \approx x'', x'' \approx true \\
 & \leadsto_{[da]} & map\ F\ [] \equiv y_2 \approx x'', x'' \approx true \\
\Longrightarrow^3 & \leadsto^2_{[da,v]} & map\ F\ [] \equiv y_2 \approx true \\
 & \leadsto_{[ona6]} & map\ F\ [] \approx [], y_2 \approx [], true \approx true \\
 & \leadsto_{[ona7]} & F \approx F'', [] \approx [], [] \approx [], y_2 \approx [], true \approx true \\
\Longrightarrow^6 & \leadsto^2_{\{F''\mapsto F\}} & [] \approx [], y_2 \approx [], true \approx true \\
\Longrightarrow^7 & \leadsto^2_{\{y_2\mapsto []\}} & true \approx true \\
 & \leadsto_{[da]} & \Box
\end{array}
$$

mit der Lösung $\{F \mapsto plus\ 0\}$. Dabei geben die $\Longrightarrow$ die Momente der Lösung der entsprechend indizierten Narrowingschritte an (im intuitiven Sinn).
Bzgl. des Systems in nicht-applikativer Schreibweise gibt es die NN-Ableitung

$$
\begin{array}{lll}
map(F,[0]) \equiv [0] & \leadsto & [@(F,0)|map(F,[])] \equiv [0] \\
 & \leadsto & @(F,0) \equiv 0 \wedge map(F,[]) \equiv [] \\
 & \leadsto_{\{F\mapsto \$plus(x_1)\}} & plus(x_1,0) \equiv 0 \wedge map(\$plus(x_1),[]) \equiv [] \\
 & \leadsto_{\{x_1\mapsto 0\}} & 0 \equiv 0 \wedge map(\$plus(0),[]) \equiv [] \\
 & \leadsto & true \wedge map(\$plus(0),[]) \equiv [] \\
 & \leadsto & map(\$plus(0),[]) \equiv [] \\
 & \leadsto & [] \equiv [] \\
 & \leadsto & true
\end{array}
$$

mit der entsprechenden Lösung $\{F \mapsto \$plus(0)\}$. Dabei entsprechen der dritte und vierte Schritt der [onv5]-Lösung und die anderen Schritte der Lösung der [ona]-Inferenzen in der Lösungsreihenfolge in der NCA-Ableitung.

Die korrespondierende Multiableitung ist von der Form

$$
\begin{array}{lll}
map(F,[0]) \equiv [0] & \looparrowright^0 & [@(F,0)|map(F,[])] \equiv [0] \\
& \looparrowright^0 & @(F,0) \equiv 0 \wedge map(F,[]) \equiv [] \\
& \looparrowright^F_{1\cdot 1, plus(0,x)\to x, \{F\mapsto \$plus(0)\}} & 0 \equiv 0 \wedge map(\$plus(0),[]) \equiv [] \\
& \looparrowright^0 & true \wedge map(\$plus(0),[]) \equiv [] \\
& \looparrowright^0 & map(\$plus(0),[]) \equiv [] \\
& \looparrowright^0 & [] \equiv [] \\
& \looparrowright^0 & true
\end{array}
$$

■

Die Beispiele legen die Formulierung des folgenden Sachverhalts nahe (im Beweis ist mit $\tau^{-1}(\overrightarrow{t_n})$ das (applikative) Tupel $\tau^{-1}(t_1)\ldots\tau^{-1}(t_n)$ gemeint).

Satz 4.4.12.

1. *Wenn es eine NN-Ableitung*

$$t_1 \looparrowright_{p_1,l_1\to r_1,\sigma_1} t_2 \looparrowright \cdots \looparrowright t_n \looparrowright_{p_n,l_n\to r_n,\sigma_n} t_{n+1}$$

gibt, dann gibt es für eine NCA-geeignete Gleichungsmenge $\tau^{-1}(t_1) \approx s$ eine NCA-Ableitung

$$
\begin{array}{lll}
\tau^{-1}(t_1) \approx s & \rightsquigarrow^* & \sigma'_1(\tau^{-1}(r_1)\overline{t_{m_1}} \approx u_1, E_1) \\
& \rightsquigarrow^* & \sigma'_2 \circ \sigma'_1(\tau^{-1}(r_2)\overline{t_{m_2}} \approx u_2, E_2) \\
& \vdots & \\
& \rightsquigarrow^* & \sigma'_n \circ \cdots \sigma'_1(\tau^{-1}(r_n)\overline{t_{m_n}} \approx u_n, E_n)
\end{array}
$$

für Tupel $\overline{t_{i_j}}$, Terme u_i und Gleichungsmengen E_i mit $1 \leq i \leq n$ und $\sigma_i(x) = \varrho(\tau(\sigma'_i(x)))$ für eine Variablenumbenennung ϱ und alle $x \in \mathcal{V}ar(t_1)$.

2. *Die Anzahl der Multischritte ist gleich der Anzahl der [onv]- und [ona]-Inferenzen.*

Beweisskizze
durch vollständige Induktion über die Länge der Ableitung.

Induktionsverankerung: $t \looparrowright_{p,l\to r,\sigma} t'$
Fallunterscheidung über die Art des Multischritts:

1.Fall:
Wenn der Schritt ein 0-Multischritt ist, dann erfolgt der Beweis identisch zum Beweis der Induktionsverankerung des Simulationssatzes 3.3.11, wobei die Rolle der [on]- hier von [ona]-Inferenzen erfüllt wird. [onv]-Inferenzen sind in dieser Ableitung nicht möglich, weil der Multischritt dann kein 0-Multischritt sein könnte. In der NCA-Ableitung haben Auftreten des Falls $R_4^{\lambda'}$ an Positionen mit @ keine Entsprechung in Form von Narrowingsequenzen; das erste Argument dieser @-Symbole ist entweder wieder ein @ oder ein Term, dessen Kopfsymbol ein Funktionssymbol ist.
Es seien also $o_1, \ldots, o_n$ mit $n \geq 0$ die Positionen der Auftreten des Falls $R_4^{\lambda'}$ und $t_i = t|_{o_i}$ für $1 \leq i \leq n$. Entsprechend seien $\tilde{o}_1, \ldots, \tilde{o}_{\tilde{n}}$ die jenigen Positionen, an denen kein @ steht. Dann gibt es wegen $Head(t) \in \mathcal{F}$ mit $t' = \tau^{-1}(t)$ eine NCA-Inferenz

$$\begin{array}{lll}
t \approx s & \rightsquigarrow^* & \sigma_1(t'|_{\tilde{o}_1} \approx u_1, E_1) \\
& \rightsquigarrow_{[ona]} & \sigma_1(t'|_{\tilde{o}_1 \cdot 1} \approx l_1|_1, \ldots, t'|_{\tilde{o}_1 \cdot n_1} \approx l_1|_{n_1}, r_1\overline{t_{m_1}} \approx u_1, E_1) \\
& \rightsquigarrow^* & \sigma_2 \circ \sigma_1(t'|_{\tilde{o}_2} \approx u_2, E_2) \\
& \rightsquigarrow_{[ona]} & \sigma_2 \circ \sigma_1(t'|_{\tilde{o}_2 \cdot 1} \approx l_2|_1, \ldots, t'|_{\tilde{o}_2 \cdot n_2} \approx l_2|_{n_2}, r_2\overline{t_{m_2}} \approx u_2, E_2) \\
& \vdots & \\
& \rightsquigarrow^* & \sigma_n \cdots \sigma_1(t'|_{\tilde{o}_{\tilde{n}}} \approx u_{\tilde{n}}, E_{\tilde{n}}) \\
& \rightsquigarrow_{[ona]} & \sigma_n \cdots \sigma_1(t'|_{\tilde{o}_{\tilde{n}} \cdot 1} \approx l_{\tilde{n}}|_1, \ldots, t'|_{\tilde{o}_{\tilde{n}} \cdot \tilde{n}} \approx l_{\tilde{n}}|_{n_{\tilde{n}}}, r_{\tilde{n}}\overline{t_{m_{\tilde{n}}}} \approx u_{\tilde{n}-1}, E_{\tilde{n}}) \\
& \rightsquigarrow^* & \vartheta \circ \sigma_n \cdots \sigma_1(r_{\tilde{n}}\overline{t_{m_{\tilde{n}}}} \approx u_{\tilde{n}-1}, E_{\tilde{n}})
\end{array}$$

unter Anwendung der Regeln $f_i(l_i|_1, \ldots, l_i|_{n_i}) \to r_i$ für $1 \leq i \leq \tilde{n}$ bei geeigneten Gleichungsmengen E_i, geeigneten Termen u_i, $t'|_{\tilde{o}_i} = f_i\overline{s_{\tilde{n}_i}}\overline{t_{m_i}}$, $l_{\tilde{n}} \to r_{\tilde{n}} = \tau^{-1}(l) \to \tau^{-1}(r)$ und $\tilde{o}_{\tilde{n}} = p$ aus dem Multischritt.
Die Entsprechung von $\vartheta \circ \sigma_n \circ \cdots \circ \sigma_1$ und σ aus dem NN-Multischritt folgt wie im Beweis von Satz 3.3.11.

2.Fall: Der Schritt ist ein F0-Multischritt.
Die dem 0-Multischritt vorangehenden c-Multischritte haben keine Entsprechung in Form von Narrowing-Inferenzen in NCA, wenn sie bis auf Variablenumbennenung keine Variablen aus t binden, weil der Term, auf den sie angewendet werden, der (partiellen) Applikation eines Konstruktors entspricht.
Es seien also $o_1, \ldots, o_n$ die Positionen derjenigen c-Multischritte, in deren jeweils erstem Schritt ein $x_i \in \mathcal{V}ar(t)$ an ein $\$c_i(\overrightarrow{y_{m_i}})$ gebunden wird ($1 \leq i \leq n,\ m_i \geq 0$).
Dann gibt es für jeden einzelnen dieser c-Multischritte eine NCA-Inferenz mit Resultat $\sigma_i(x_i\overline{s_j} \approx c_i\overline{t_{m_i}}\overline{v_j}, E_i)$, auf die eine [dv]-Inferenz mit Resultat $\vartheta_i \circ \sigma_i(s_1 \approx v_1, \ldots, s_j \approx v_j, E_i)$ mit $\vartheta_i = \{x_i \mapsto \overline{t_{m_i}}\}$ angewendet werden kann. Die Bindung der Variablentupel $\overrightarrow{y}_{m_i}$ an $\tau(\overline{t_{m_i}})$ erfolgt im F0-Multischritt, weil die $\overline{t_{m_i}}$ Teilterme linker Regelseiten sein müssen und die Bindung bei Traversierung der definierenden Bäume geschieht (da in den vorausgegangen Narrowingsequenzen stets *alle* passenden Regeln ausprobiert werden, gibt es insbesondere solche $\tau(\overline{t_{m_i}})$, die Instanzen von Teiltermen der entsprechenden Knoten sind). Mit demselben Argument gibt es dann eine NCA-Inferenz mit Resultat $\psi(t|_p \approx u, E)$ für eine Substitution ψ, auf das der abschließende 0-Multischritt wie im ersten Fall angewendet werden kann.

3.Fall: Der Schritt ist ein F-Multischritt.
Wegen der Konstruktorbasiertheit der Regeln (die insbesondere kein @ enthalten) ist

$$t_1|_p = @_{n-1}(\cdots(@_1(x, t'_1), t'_2), \cdots, t'_{n-1}).$$

Da es sich um eine erfolgreiche Ableitung handelt, muß $x \in \mathcal{V}ar$ oder $x = \$g(\overrightarrow{g_k})$ für ein $g \in \mathcal{F} \cup \mathcal{C}$ mit $k \geq 0$ sein.

- Es ist $x \in \mathcal{V}ar$.

  Die ersten $n-1$ Schritte dieses Multischritts führen dann zu einem Term $\vartheta \circ \psi(t_1[g(\overrightarrow{x_k}, \overrightarrow{t'_{n-1}})]_p)$ mit einer Substitution ψ, die von λ' im ersten Schritt des Multischritts "bis zur Position p" berechnet wurde, und $\vartheta = \{x \mapsto \$g(\overrightarrow{x_k})\}$, für den abschließend an der Position p Narrowing mit der Regel $g(\overrightarrow{l_{k+n-1}}) \to r$ (nach Definition der F-Multischritte ist $l := g(\overrightarrow{l_{k+n-1}})$) und der Substitution $\eta = mgu\big(g(\overrightarrow{l_{k+n-1}}), \varphi(g(\overrightarrow{x_k}, \overrightarrow{t'_{n-1}}))\big)$ durchgeführt werden kann, wobei φ bei Traversieren der DTs die $\overrightarrow{x_k}$ an $\overrightarrow{l_k}$ bindet. Nach Konstruktion ist $\sigma = \eta \circ \varphi \circ \vartheta \circ \psi[\mathcal{V}ar(t_1)]$.

  Die Behandlung der c-Multischritte, die den Anfang dieses Schritts bilden, erfolgt identisch zum zweiten Fall. Wie dort gibt es dann eine NCA-Inferenz (die insb. [onv]- und [dv]-Inferenzen enthält, die durch die einzelnen c-Multischritte simuliert werden) mit Resultat $\psi'\big((\tau^{-1}(t_1)|_p)\overline{s_i} \approx u_1, E_1\big)$ für ein Tupel $\overline{s_i}$, einen Term u_1 und eine Gleichungsmenge E_1 mit $\tau(\psi'(x)) = \psi(x)$ für alle $x \in \mathcal{V}ar$. Da auch das leftmost-innermost Symbol von $\psi'(\tau^{-1}(t_1|_p))$ (das ist ein applikativer Term) eine Variable ist, sei $\psi'\big((\tau^{-1}(t_1)|_p)\overline{s_i}\big) = x\overline{t_j}\overline{s'_i}$ für geeignete Tupel $\overline{t_j}$, $\overline{s'_i}$ und eine Variable x. Dabei kann $\overline{t_j} = \tau^{-1}(\overrightarrow{t'_{n-1}})$ bis auf Variablenumbenennung angenommen werden, weil der korrespondierende NN-Schritt möglich ist. Unter Verwendung derselben Regel wie in jenem Schritt ist dann eine [onv]-Inferenz mit Resultat

  $$\vartheta'(t'_1 \approx l_{k+1}, \ldots, t'_{n-1} \approx l_{k+n-1}, r\overline{s'_i} \approx u_1, E_1)$$

  für $\vartheta' = \{x \mapsto g\overline{l_k}\}$ möglich.

  Da der n-te Schritt des Multischritts unter Anwendung der Regel $g(\overrightarrow{l_{k+n-1}}) \to r$ stattfindet, müssen die Paare $\langle\vartheta'(t'_i), \vartheta'(l_{k+i})\rangle$ für $1 \leq i < n$ syntaktisch, d.h. modulo dem leeren TES $\emptyset$, unifizierbar sein. Das bedeutet, daß die Unifikation unter Anwendung der Inferenzregeln [da], [dv] und [v] erfolgen kann, wobei bei Anwendung der [dv]-Regel auf Terme $x\overline{s_n}$ stets $n = 0$ ist (das ist ein entscheidender Punkt, weil andernfalls ein Lookahead auf die nächsten NCA-Inferenzen nötig wäre und damit kein Induktionsbeweis geführt werden könnte).

  Das Resultat dieser Ableitung ist $\eta' \circ \vartheta'(r\overline{s'_i} \approx u_1, E_1)$.

  Nach Konstruktion ist dann $\eta' \circ \vartheta' \circ \psi'(x) = \tau^{-1}(\eta \circ \vartheta \circ \psi(x))$ für alle $x \in \mathcal{V}ar(t_1)$ bis auf Variablenumbenennung, denn es ist insbesondere $\eta \circ \vartheta(\$g(\overrightarrow{x_k})) = \$g(\overrightarrow{l_k})$ nach Konstruktion von η.

- Es ist $x = \$g(\overrightarrow{g_k})$.

  Der Beweis funktioniert dann analog. Der Unterschied ist der, daß anstelle der [onv]-Inferenz eine [ona]-Inferenz durchgeführt wird und damit $\vartheta = \vartheta' = \emptyset$ ist.

Induktionsschritt:
Der Induktionsschritt erfolgt wie im Beweis von Satz 3.3.11 über eine Fallunterscheidung über die relative Position der Narrowingschritte zueinander.

Die Aussage über die Anzahl der Schritte folgt dann aus der Konstruktion der NCA-Ableitung im Beweis.

□

Da NCA korrekt und vollständig ist, kann diese Simulation als ein Korrektheitsbeweis für Warren's Methode angesehen werden (s.S. 93):

Satz 4.4.13.
Warren's Methode ist korrekt in dem Sinn, daß jede Lösung einer Anfrage bzgl. eines (mit Warren's Methode) transformierten Systems auch Lösung bzgl. des entsprechenden applikativen Systems ist.

Beweis
Folgt aus der Simulationsbeziehung.

□

4.5 Zur Simulation von NCA durch NN

Die wechselseitigen Simulationsbeziehung beider Kalküle kann nicht wie im Fall erster Ordnung durch Reduktion der "Rückrichtung" vorgenommen werden, weil das entscheidende Argument dort darauf beruhte, daß OINC-Ableitungen LOI und damit eindeutig sind. Eine solche Eigenschaft ist für NCA nicht einfach nachzuweisen.
Es bietet sich deshalb an, NCA-Inferenzen durch NN- oder OINC-Ableitungen zu simulieren.
Für die NCA-Inferenzregeln [da], [dv] und [v] können entsprechende OINC-Ableitungen angegeben werden, die durch die Simulierbarkeit beider Kalküle in direktem Bezug zu NN-Ableitungen stehen. Dieses wird in den Lemmata 4.5.2, 4.5.4 und 4.5.6 vorgeführt.
Die Simulation wird also indirekt gezeigt: Die angegebenen NCA-Inferenzen bzgl. $\mathcal{R}$ werden durch OINC-Ableitungen bzgl. $\mathcal{R}^{@}$ simuliert, und diese werden – da $\mathcal{R}^{@}$ nach Voraussetzung induktiv-sequentiell ist – durch NN-Ableitungen simuliert.

Für die NCA-Inferenzen [ona] und [onv] ist das sehr viel schwieriger:

1. Die in beiden Inferenzen entstehende Gleichung $r\overline{t_m}$ hat keine Entsprechung in OINC (bzgl. $\mathcal{R}^@$ *und* $\mathcal{R}$). Das liegt daran, daß OINC für einen Term $f\overline{s_n}\overline{t_m}$ zunächst m [on]- oder [d]-Schritte durchführt und damit das Tupel $\overline{t_m}$ "auseinandergezogen" wird. Da es schwierig ist, Voraussagen über das Verhalten von OINC zu machen, bis OINC die Komponenten dieses Tupels "erreicht" (d.h. sie sind erstes Element der Gleichungsmenge), ist die Simulation von NCA durch OINC (bzgl. $\mathcal{R}^@$) und damit durch NN nicht offensichtlich: Eine erfolgreiche NCA-Ableitung, die mit einer [ona]-Inferenz $f\overline{s_n}\overline{t_m} \approx t, E \leadsto_{[ona]} s_1 \approx u_1, \ldots, s_n \approx u_n, r\overline{t_m} \approx t, E$ für eine frische Regel $f\overline{u_n} \to r$ beginnt, wird schließlich zu $\sigma(r\overline{t_m} \approx t, E)$ für eine Substitution σ abgeleitet. Dann wird entschieden, wie mit dem Tupel $\overline{t_m}$ verfahren wird, welche Inferenz also gewählt wird. In OINC (bzgl. $\mathcal{R}^@$ muß diese Entscheidung früher erfolgen: $@_m(\cdots @_1(f(\tau(s_1), \ldots, \tau(s_n)), \tau(t_1)) \cdots, \tau(t_m))$ wird (ggf. in Abhängigkeit von t) in m [d]- oder [on]-Schritten so abgeleitet, bis $f(\tau(s_1), \ldots, \tau(s_n))$ als erste Gleichungsmenge der Gleichung auftaucht. Dann erst kann der gleiche Narrowingschritt wie in NCA durchgeführt werden.

   Im Beweis der Vollständigkeit von NCA [NMI95] wird gezeigt, daß für jede erfolgreiche OINC-Ableitung eine erfolgreiche NCA-Ableitung existiert. Es wird auch gezeigt, wie diese konkret aussieht; allerdings kann sie aus den genannten Gründen nicht *direkt* mit der OINC-Ableitung vergleichen werden.

   Solche Situationen (d.h. Terme $r\overline{t_m}$) treten nur dann auf, wenn auf rechten Regelseiten partiell applizierte Funktionen zugelassen werden. Ist das nicht der Fall, kann eine Simulation angegeben werden (s. S. 119; Bsp. 4.5.7).

2. Die Simulation von OINC-Ableitungen bzgl. ATES durch OINC-Ableitungen bzgl. der entsprechenden TES mit den durch die Warren'sche Methode neu erzeugten Regeln ist für die [on]-Inferenz schwierig: Wenn bzgl. eines ATES eine Gleichung $f\overline{s_n}\overline{t_m} \approx t$ mit $Arity(f) = n$ entsteht, dann gibt es (exemplarisch für $m = 1$) die Ableitung

$$f\overline{s_n}t_1 \approx t \leadsto_{[on]} f\overline{s_n} \approx g\overline{u_k}, t_1 \approx u_{k+1}, r \approx t$$

   für eine frische Regel $g\overline{u_{k+1}} \to r$. Bzgl. des explizit applikativen TES (mit den neuen Regeln) gibt es dann die Ableitung

$$\begin{aligned} @(f(\tau(\overrightarrow{s_n}), \tau(t_1)) \approx \tau(t) \quad \leadsto_{[on]} \quad & f(\tau(\overrightarrow{s_n}) \approx \$h(\overrightarrow{x_l}), \\ & \tau(t_1) \approx x_{l+1}, \\ & [\$]h(\overrightarrow{x_{l+1}}) \approx t, \end{aligned}$$

   wobei $\overrightarrow{v_j}$ für $v_1, \ldots, v_j$ steht und $@(\$h(\overrightarrow{x_l}), x_{l+1}) \to [\$]h(\overrightarrow{x_{l+1}})$ die verwendete Regel ist (mit dem optionalen \$-Zeichen, wenn $l + 1 < Arity(h)$ ist und ohne es, falls $l + 1 = Arity(h)$ ist).

Auch in diesem Fall ist der direkte Zusammenhang zwischen beiden Ableitungen nicht ohne weiteres herzustellen, weil in der ersten Ableitung die rechte Regelseite r direkt und in der zweiten erst nach der Lösung der anderen (weiter links stehenden) Gleichungen erzeugt wird. Gelänge diese Zuordnung, so wäre zunächst die "Vollständigkeit" der Warren'schen Transformation nachgewiesen (für jede erfolgreiche Ableitung eines vollständigen Kalküls für Anfragen höherer Ordnung gibt es eine erfolgreiche Ableitung eines Kalküls erster Ordnung in Verbindung mit der Warren'schen Methode). Der Nachweis des umgekehrten Sachverhalts wurde bereits als "Korrektheit" von Warren's Transformation gedeutet.

Beide Beweise würden aber noch nicht die Simulation von NCA durch NN liefern. Der Vollständigkeitsbeweis von NCA (Kapitel 5 in [NMI95]) basiert zwar auf dem Nachweis, daß für jede erfolgreiche OINC-Ableitung eine erfolgreiche NCA-Ableitung mit derselben Lösung existiert. Zwischen diesen Ableitungen besteht allerdings keine gegenseitige Simulationsbeziehung (die OINC-Ableitungen werden in andere OINC-Ableitungen transformiert, die im Verlauf der Berechnung der Lösung einer Anfrage niemals erzeugt würden).

3. Die Simulation von NCA durch NN ist ebenfalls schwierig, weil Voraussagen über das Aussehen der Ableitungen nicht leicht zu treffen sind (das ist auch der Grund dafür, daß bei der Simulation von OINC durch NN auf eine direkte Angabe der similaren Ableitungen verzichtet wurde und der "Umweg" über die Eindeutigkeit der Lösungen gewählt wurde).

Es wird deshalb auf eine die Angabe von NCA- durch NN simulierte Ableitungen verzichtet. Stattdessen wird nur für die Inferenzregeln [da], [dv] und [v] eine Simulation durch OINC bzgl. $\mathcal{R}^{@}$ *und damit durch NN* angegeben. Für die Regeln [ona] und [onv] wird gezeigt, daß das Verhalten "so ähnlich" wie das von NN ist (man beachte, daß nur einzelne Schritte und nicht gesamte Ableitungen in Bezug zueinander gesetzt werden).

Für die Simulation der Dekompositionsregeln von NCA durch OINC wird das folgende Lemma benötigt:

Lemma 4.5.1.
In jeder erfolgreichen OINC-Ableitung in flachen applikativen Systemen mit ausschließlich trues als initialen rechten Gleichungsseiten gilt für alle Teilterme rechter Gleichungsseiten der Form $f\overline{s_n}$ mit $n > 0$:

1. $f \notin \mathcal{V}ar$
2. $n < Arity(f)$, *falls* $f \in \mathcal{F}$ *und*
3. $n \leq Arity(f)$, *falls* $f \in \mathcal{C}$.

Beweis
durch Induktion über die Länge der Ableitung:

Die erste Behauptung folgt aus der Tatsache, daß linke Regelseiten Muster sind, also insbesondere kein Teilterm als Kopfsymbol eine Variable besitzt. Außerdem werden rechtsnormale initiale Gleichungsmengen vorausgesetzt.

Die Induktionsverankerung (leere Ableitung) ist trivial.
Es sei $f\overline{s_n} \approx t, E$ eine inferierte Gleichungsmenge. Dann führt

- mit $f \in \mathcal{F} \cup \mathcal{V}ar$ und $Arity(f) = m$ eine [on]-Inferenz zu der Gleichungsmenge $f\overline{s_{n-1}} \approx g\overline{l_{m-1}}, s_n \approx l_m, r \approx t$ bei Anwendung einer Regel $g\overline{l_m} \to r$.

  Die Behauptung gilt dann wegen $m - 1 < m$ für $f\overline{s_{n-1}} \approx g\overline{l_{m-1}}$, weil die l_i keine Funktionssymbole enthalten und in linken Regelseiten keine "überapplizierten" Konstruktoren stehen (allerdings vollständig applizierte, deshalb benötigt man die Gleichheit in der dritten Behauptung). Für $s_n \approx l_m$ gilt sie mit derselben Argumentation, und für $r \approx t$ gilt sie nach Induktionsvoraussetzung;

- mit $f \in \mathcal{F} \cup \mathcal{C} \cup \mathcal{V}ar$ und $t = g\overline{t_m}$ für $m > 0$ eine [d]-Inferenz zu der Gleichungsmenge $f\overline{s_{n-1}} \approx g\overline{t_{m-1}}, s_n \approx t_m$.

  Die Behauptung ergibt sich dann aus der Induktionsvoraussetzung (die Bedingungen gelten für alle Teilterme rechter Gleichungsseiten);

Für inferierte Gleichungsmengen $x \approx t, E$ mit $t \notin \mathcal{V}ar$ oder $t \approx x, E$ mit $x \in \mathcal{V}ar$ in beiden Fällen führt eine [v]-Inferenz zu $\{x \mapsto t\}(E)$. Die Behauptung gilt für $\{x \mapsto t\}(E)$, denn wegen der Linkslinearität der Regeln und der Rechtsnormalität initialer Gleichungen kann x auf keiner rechten Gleichungsseite in E vorkommen.

□

Damit können nun Simulationen für [da], [dv] und [v] konstruiert werden:

Lemma 4.5.2.
Wenn es eine NCA-Ableitung

$$A : f\overline{s_n} \approx f\overline{t_n}, E \leadsto_{[da]} s_1 \approx t_1, \ldots , s_n \approx t_n, E$$

bzgl. eines flachen applikativen TES $\mathcal{R}$ gibt, dann gibt es eine OINC-Ableitung

$$B : \tau(f\overline{s_n}) \approx \tau(f\overline{t_n}), \tau(E) \leadsto_{[d]} \tau(s_1) \approx \tau(t_1), \ldots , \tau(s_n) \approx \tau(t_n), \tau(E)$$

bzgl. $\mathcal{R}^@$.

Beweis durch vollständige Induktion über n:
Wenn $n = 0$ ist, dann hat A die Form $f \approx f, E \leadsto_{[d]} E$ und B die Form $\tau(f) \approx \tau(f), \tau(E) \leadsto_{[d]} \tau(E)$.

Wenn $n > 0$ ist, so hat B

- für $n = \mathit{Arity}(f)$ die Form
$$\begin{array}{rl} & f(\tau(s_1),\ldots,\tau(s_n)) \approx f(\tau(t_1),\ldots,\tau(t_n)),\tau(E) \\ \rightsquigarrow_{[d]} & \tau(s_1)\approx\tau(t_1),\ldots,\tau(s_n)\approx\tau(t_n),\tau(E), \end{array}$$
denn gemäß Lemma 4.5.1 ist in diesem Fall $f \in \mathcal{C}$;
- für $n < \mathit{Arity}(f)$ die Form
$$\begin{array}{rl} & \$f(\tau(s_1),\ldots,\tau(s_n)) \approx \$f(\tau(t_1),\ldots,\tau(t_n)),\tau(E) \\ \rightsquigarrow_{[d]} & \tau(s_1)\approx\tau(t_1),\ldots,\tau(s_n)\approx\tau(t_n),\tau(E); \end{array}$$
- und der Fall $n > \mathit{Arity}(f)$ kann gemäß Lemma 4.5.1 nicht auftreten.

□

Beispiel 4.5.3.
Bei gegebenem ATES, das nur aus den Regeln für die strikte Gleichheit besteht, gibt es für die Anfrage $c\ x\ y \equiv c\ a\ b \approx \mathit{true}$ die NCA-Inferenz

$$\begin{array}{lll} c\ x\ y \equiv c\ a\ b \approx \mathit{true} & \rightsquigarrow_{[ona]} & c\ x\ y \approx c\ x_1\ y_1, c\ a\ b \approx c\ x_2\ y_2, \\ & & x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow_{[da]} & x \approx x_1, y \approx y_1, c\ a\ b \approx c\ x_2\ y_2, \\ & & x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow^2_{\{x_1\mapsto x, y_1\mapsto y\}} & c\ a\ b \approx c\ x_2\ y_2, \\ & & x \equiv x_2 \wedge y \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow_{[da]} & a \approx x_2, b \approx y_2, \\ & & x \equiv x_2 \wedge y \equiv y_2 \\ & \rightsquigarrow^2_{\{x_2\mapsto a, y_2\mapsto b\}} & x \equiv a \wedge y \equiv b \approx \mathit{true} \\ & \rightsquigarrow^+_{\{x\mapsto a, y\mapsto b\}} & \square. \end{array}$$

Für das TES $\mathcal{R} = \{@(\$c, x) \to \$c(x), @(\$c(x), y) \to c(x,y)\}$ gibt es bzgl. $\mathcal{R}^@_\equiv$ die OINC-Ableitung

$$\begin{array}{lll} c(x,y) \equiv c(a,b) \approx \mathit{true} & \rightsquigarrow_{[on]} & c(x,y) \approx c(x_1,y_1), c(a,b) \approx c(x_2,y_2), \\ & & x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow_{[d]} & x \approx x_1, y \approx y_1, c(a,b) \approx c(x_2,y_2), \\ & & x_1 \equiv x_2 \wedge y_1 \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow^2_{\{x_1\approx x, y_1\approx y\}} & c(a,b) \approx c(x_2,y_2), \\ & & x \equiv x_2 \wedge y \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow_{[d]} & a \approx x_2, b \approx y_2, \\ & & x \equiv x_2 \wedge y \equiv y_2 \approx \mathit{true} \\ & \rightsquigarrow^2_{\{x_2\mapsto a, y_2\mapsto b\}} & x \equiv a \wedge y \equiv b \approx \mathit{true} \\ & \rightsquigarrow^+_{\{x\mapsto a, y\mapsto b\}} & \square. \end{array}$$

In der korrespondierenden NN-Ableitung

$$\begin{array}{rll} c(x,y) \equiv c(a,b) & \rightsquigarrow & x \equiv a \wedge y \equiv b \\ & \rightsquigarrow_{\{x \mapsto a\}} & true \wedge y \equiv b \\ & \rightsquigarrow^2_{\{y \mapsto b\}} & true \end{array}$$

werden diese Schritte implizit durchgeführt.

■

Lemma 4.5.4.
Wenn es eine NCA-Ableitung

$$A : x\overline{s_n} \approx f\overline{t_m}\,\overline{u_n}, E \rightsquigarrow_{[dv]} \vartheta(s_1 \approx u_1, \ldots, s_n \approx u_n, E)$$

bzgl. eines flachen applikativen TES $\mathcal{R}$ mit $\vartheta = \{x \mapsto f\overline{t_m}\}$ gibt, dann gibt es eine OINC-Ableitung

$$B : \tau(x\overline{s_n}) \approx \tau(f\overline{t_m}\,\overline{u_n}), \tau(E) \rightsquigarrow^* \vartheta'\big(\tau(s_1 \approx u_1, \ldots, s_n \approx u_n, E)\big)$$

bzgl. $\mathcal{R}^@$ mit $\vartheta' \restriction_{\mathcal{V}ar(x\overline{s_n}) \cup \mathcal{V}ar(f\overline{t_m}\,\overline{u_n}) \cup \mathcal{V}ar(E)} = \{x \mapsto \$f(\tau(t_1), \ldots, \tau(t_m))\}$.

Beweis
Es werden zwei Fälle unterschieden:
1. Fall: $m + n = Arity(f)$.
Dann ist gemäß Lemma 4.5.1 $f \in \mathcal{C}$, d.h. es gibt eine erfolgreiche Ableitung in $\mathcal{R}^@$ der Form

$$\begin{array}{ll}
& @_n(\cdots @_1(x, \tau(s_1)) \cdots, \tau(s_n)) \\
& \quad \approx f(\tau(t_1), \ldots, \tau(t_m), \tau(u_1), \ldots, \tau(u_n)), E \\
\rightsquigarrow_{[on]} & @_{n-1}(\cdots @_1(x, \tau(s_1)) \cdots, \tau(s_{n-1})) \approx \$f(x_1^{(1)}, \ldots, x_{m+n-1}^{(1)}), \\
& \tau(s_n) \approx x_{m+n}^{(1)}, f(x_1^{(1)}, \ldots, x_{m+n}^{(1)}) \\
& \quad \approx f(\tau(t_1), \ldots, \tau(t_m), \tau(u_1), \ldots, \tau(u_n)), E \\
\rightsquigarrow_{[on]} & @_{n-2}(\cdots @_1(x, \tau(s_1)) \cdots, \tau(s_{n-2})) \approx \$f(x_1^{(2)}, \ldots, x_{m+n-2}^{(2)}), \\
& \tau(s_{n-1}) \approx x_{m+n-1}^{(2)}, \$f(x_1^{(2)}, \ldots, x_{m+n-1}^{(2)}) \approx \$f(x_1^{(1)}, \ldots, x_{m+n-1}^{(1)}), \\
& \tau(s_n) \approx x_{m+n}^{(1)}, \\
& f(x_1^{(1)}, \ldots, x_{m+n}^{(1)}) \approx f(\tau(t_1), \ldots, \tau(t_m), \tau(u_1), \ldots, \tau(u_n)), E \\
\rightsquigarrow^{n-2}_{[on]} & x \approx \$f(x_1^{(n)}, \ldots, x_m^{(n)}), \\
& \tau(s_1) \approx x_{m+1}^{(n)}, \$f(x_1^{(n)}, \ldots, x_{m+1}^{(n)}) \approx \$f(x_1^{(n-1)}, \ldots, x_{m+1}^{(n-1)}), \\
& \tau(s_2) \approx x_{m+2}^{(n-1)}, \$f(x_1^{(n-1)}, \ldots, x_{m+2}^{(n-1)}) \approx \$f(x_1^{(n-2)}, \ldots, x_{m+2}^{(n-2)}), \\
& \quad \vdots \\
& \tau(s_{n-1}) \approx x_{m+n-1}^{(2)}, \$f(x_1^{(2)}, \ldots, x_{m+n-1}^{(2)}) \approx \$f(x_1^{(1)}, \ldots, x_{m+n-1}^{(1)}), \\
& \tau(s_n) \approx x_{m+n}^{(1)}, f(x_1^{(1)}, \ldots, x_{m+n}^{(1)}) \\
& \quad \approx f(\tau(t_1), \ldots, \tau(t_m), \tau(u_1), \ldots, \tau(u_n)), E \\
\rightsquigarrow^{i}_{[d,v]} & f(x_1^{(n)}, \ldots, x_m^{(n)}, \tau(s_1), \ldots, \tau(s_n)) \\
& \quad \approx f(\tau(t_1), \ldots, \tau(t_m), \tau(u_1), \ldots, \tau(u_n)), E \\
\rightsquigarrow^{j}_{[d,v]} & \vartheta'(\tau(s_1) \approx \tau(u_1), \ldots, \tau(s_n) \approx \tau(u_n), E)
\end{array}$$

- unter Anwendung von

$$@(\$f(x_1^{(1)}, \dots, x_{m+n-1}^{(1)}), x_{m+n}^{(1)}) \to f(x_1^{(1)}, \dots, x_{m+n}^{(1)})$$

im ersten [on]-Schritt und von

$$@(\$f(x_1^{(i)}, \dots, x_{m+n-i}^{(i)}), x_{m+n+1-i}^{(i)}) \to \$f(x_1^{(i)}, \dots, x_{m+n+1-i}^{(i)})$$

im $i = 2., 3., \dots, n.$ [on]-Schritt,

- mit $\vartheta' \restriction_{\mathcal{V}ar(x\overline{s_n}) \cup \mathcal{V}ar(f\overline{t_m}\overline{u_n}) \cup (E)} = \{x \mapsto \$f(t_1, \dots, t_m)\}$, denn die Variablen $x_k^{(l)}$ sind alle frisch,

- $i = (n^2+5n+2)/2$, denn ein [v]-Schritt bindet x, n [v]-Schritte binden die Gleichungen mit $\tau(s_i)$ auf der linken Seite, n [d]-Schritte dekomponieren die Gleichungen der Form $\$f(\dots) \approx \$f(\dots)$ bzw. $f(\dots) \approx f(\dots)$, und für diese Gleichungen müssen insgesamt $\sum_{i=m+1}^{m+n} i = n(n+1)/2$ Variablenbindungen durchgeführt werden, d.h. insgesamt werden $(n^2+3n+2)/2$ [v]- und n [d]-Inferenzen durchgeführt.

- $j = m + 1$ für eine Dekomposition und m [v]-Schritte.

2. Fall: $m + n < \mathit{Arity}(f)$
Dann ist $t = \$f(t_1, \dots, t_m, u_1, \dots, u_n)$.
Die resultierende Sequenz hat das gleiche Aussehen wie die des ersten Falls, mit dem Unterschied, daß im ersten Schritt die Regel

$$@(\$f(x_1^{(1)}, \dots, x_{m+n-1}^{(1)}), x_{m+n}^{(1)}) \to \$f(x_1^{(1)}, \dots, x_{m+n}^{(1)})$$

anstelle von

$$@(\$f(x_1^{(1)}, \dots, x_{m+n-1}^{(1)}), x_{m+n}^{(1)}) \to f(x_1^{(1)}, \dots, x_{m+n}^{(1)})$$

verwendet wird.

Der Fall $m + n > \mathit{Arity}(f)$ kann nach Lemma 4.5.1 nicht auftreten.

□

Für die Simulation einer [dv]-Inferenz für eine linke Gleichungsseite $x\overline{s_n}$ sind also n [on]-Inferenzen bzw. NN-Schritte nötig.

Beispiel 4.5.5.
$\mathcal{R} = \{f\ (c\ a\ b\ d) \to a\}$ sei ein ATES. Dann gibt es bzgl. $\mathcal{R}_\equiv$ für die Anfragen (1) $f(x\ y) \equiv a \approx true$, (2) $f(x\ y\ z) \equiv a \approx true$, und (3) $f(x\ y\ z\ z') \equiv a \approx true$ die NCA-Ableitungen

$$\begin{array}{rll}
f\ (x\ y) \equiv a \approx true & \leadsto_{[ona]} & f(x\ y) \approx a, a \approx a, true \approx true \\
& \leadsto_{[ona]} & x\ y \approx c\ a\ b\ d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{x \mapsto c\ a\ b\}} & y \approx d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{y \mapsto d\}} & a \approx a, true \approx true \\
& \leadsto^2 & \Box,
\end{array}$$

$$\begin{array}{rll}
f\ (x\ y\ z) \equiv a \approx true & \leadsto_{[ona]} & f(x\ y\ z) \approx a, a \approx a, true \approx true \\
& \leadsto_{[ona]} & x\ y\ z \approx c\ a\ b\ d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{x \mapsto c\ a\}} & y \approx b, z \approx d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{y \mapsto b\}} & z \approx d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{z \mapsto d\}} & a \approx a, true \approx true \\
& \leadsto^2 & \Box \quad und
\end{array}$$

$$\begin{array}{rll}
f\ (x\ y\ z\ z') \equiv a \approx true & \leadsto_{[ona]} & f(x\ y\ z\ z') \approx a, a \approx a, true \approx true \\
& \leadsto_{[ona]} & x\ y\ z\ z' \approx c\ a\ b\ d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{x \mapsto c\}} & y \approx a, z \approx b, z' \approx d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{y \mapsto a\}} & z \approx b, z' \approx d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{z \mapsto b\}} & z' \approx d, a \approx a, true \approx true \\
& \leadsto_{[dv],\{z' \mapsto d\}} & a \approx a, true \approx true \\
& \leadsto^2 & \Box.
\end{array}$$

Für $\mathcal{S} = \{f(c(a,b,d)) \to a\}$ gibt es bzgl. $\mathcal{S}^@_\equiv$ die NN-Ableitungen

$$\begin{array}{rll}
f(@(x,y)) \equiv a & \leadsto_{\{x \mapsto \$c(x_1,y_1)\}} & f(c(x_1,y_1,y)) \equiv a \\
& \leadsto_{\{x_1 \mapsto a, y_1 \mapsto b, y \mapsto d\}} & a \equiv a \\
& \leadsto & true,
\end{array}$$

$$\begin{array}{rll}
f(@(@(x,y),z) \equiv a & \leadsto_{\{x \mapsto \$c(x_1)\}} & f(@(\$c(x_1,y),z)) \equiv a \\
& \leadsto & f(c(x_1,y,z)) \equiv a \\
& \leadsto_{\{x_1 \mapsto a, y \mapsto b, z \mapsto d\}} & a \equiv a \\
& \leadsto & true \quad und
\end{array}$$

$$\begin{array}{rll}
f(@(@(@(x,y),z),z')) \equiv a & \leadsto_{\{x \mapsto \$c\}} & f(@(@(\$c(y),z),z')) \equiv a \\
& \leadsto & f(@(\$c(y,z),z')) \equiv a \\
& \leadsto & f(c(y,z,z')) \equiv a \\
& \leadsto_{\{y \mapsto a, z \mapsto b, z' \mapsto d\}} & a \equiv a \\
& \leadsto & true
\end{array}$$

■

Lemma 4.5.6.
Wenn es eine NCA-Sequenz der Form $t \approx x, E \rightsquigarrow_{[v]} \{x \mapsto t\}(E)$ bzgl. eines flachen applikativen Systems $\mathcal{R}$ gibt, dann gibt es eine OINC-Sequenz $\tau(t) \approx x, \tau(E) \rightsquigarrow_{[v]} \{x \mapsto \tau(t)\}(\tau(E))$ bzgl. $\mathcal{R}^{@}$.

Beweis
Offensichtlich.

□

Wie weiter oben erläutert wurde, ist die Simulation der Narrowing-Inferenzen nicht unproblematisch, wenn auf rechten Regelseiten partiell applizierte Funktionen zugelassen werden (andernfalls ist die Simulation möglich, s. S. 119; Bsp. 4.5.7). Es wird hier deshalb nur auf eine gewisse Ähnlichkeit von NN und NCA in den entsprechenden Situationen hingewiesen (die der "Rückrichtung", d.h. der Simulation von NN durch NCA in Kapitel 4.4 entspricht). Die vorläufige Diskussion der Komplexitäten erfolgt hier, weil sie auch für die anderen NCA-Inferenzen in diesem Kapitel durchgeführt wurde.

Zur Erinnerung: Eine [ona]-Inferenz ist von der Form

$$f\overline{s_n}\overline{t_m} \approx t, E \rightsquigarrow_{[ona]} s_1 \approx u_1, \ldots, s_n \approx u_n, r\overline{t_m} \approx t, E,$$

für $t \notin \mathcal{V}ar$ und eine Regel $f\overline{u_n} \to r$.
Wenn eine solche Inferenz für einen Term $f\overline{s_n}\overline{t_m}$ mit $n = Arity(f)$ durchgeführt wird, dann werden in der folgenden Ableitungsfolge zunächst die einzelnen Argumente s_i von f mit den Argumenten u_i der linken Regelseite modulo $\mathcal{R}$ unifiziert. Die resultierende Substitution σ dieser Berechnung wird dann auf r angewendet, und da dieses Ergebnis eine partiell applizierte Funktion sein muß (sonst wäre die Ableitung nicht erfolgreich), werden die restlichen Terme $t_1, \ldots, t_m$ als Argumente für eben diese partiell applizierte Funktion eingesetzt. Das Ergebnis vor einer Auswerung von r ist damit $\sigma(r\overline{t_m})$.

Wie verhält sich nun NN bzgl. $\mathcal{R}^{@}$ in einer solchen Situation? Es ist $\tau(f\overline{s_n}\overline{t_m}) = @_m(\cdots @_1(f(\tau(s_1), \ldots, \tau(s_n)), \tau(t_1)) \cdots, \tau(t_m))$ wegen $n = Arity(f)$.
NN führt nun zunächst das Narrowing für $f(\tau(s_1), \ldots, \tau(s_n))$ durch, indem die einzelnen Argumente modulo $\mathcal{R}^{@}$ mit dem Muster eines DT unifiziert werden. Das Resultat dieser Narrowingschritte ist dann – wie für NCA – die partiell applizierte Funktion r, und als Ergebnis ergibt sich für eine der NCA-Ableitung entsprechenden Ableitung der Term $@_m(\cdots @_1(\sigma(\tau(r))), \tau(t_1)) \cdots, \tau(t_m))$ mit $\tau^{-1}\big(@_m(\cdots @_1(\sigma(\tau(r)), \tau(t_1)) \cdots, \tau(t_m))\big) = \sigma(r\overline{t_m})$.

An dieser Stelle wird deutlich, daß das Verhalten von NCA i.a. ungünstiger als das von NN sein wird, weil NCA wie OINC die anzuwendende Regel "zu früh" rät. Die Konsequenzen dieses Verhaltens wurden bereits geschildert.

Eine ähnliche Betrachtung kann nun für die [onv]-Inferenz angestellt werden. Die NCA-Ableitung ist dann von der Form

$$x\overline{s_n}\overline{t_m} \approx t, E \rightsquigarrow_{[onv]} \{x \mapsto f\overline{u_k}\}(s_1 \approx v_1, \ldots, s_n \approx v_n, r\overline{t_m} \approx t, E)$$

für $t \notin \mathcal{V}ar$ und eine frische Regel $f\overline{u_k v_n} \to r$ mit $n > 0$.
Eine Implementierung von NCA wird Informationen über die Stelligkeiten der Funktionssymbole zur Verfügung haben. Damit müssen bei k definierten Funktionssymbolen der Stelligkeiten a_k , für die jeweils b_k Regeln existieren, $\sum_{i=1}^{k} b_i$ Regeln (d.h. alle) ausprobiert werden. Außerdem müssen für jede Regel der Stelligkeit a_i die a_i Partitionierungen in zwei Summanden (von denen der zweite gleich Null sein darf) berechnet und ausprobiert werden, d.h. also, daß tatsächlich $\sum_{i=1}^{k} a_i \cdot b_i$ verschiedene Regeln betrachtet werden müssen. Der Nachteil des frühen Ratens der Regeln wurde bereits diskutiert.

Im Fall von NN ergibt sich folgendes Bild: Es ist
$\tau(x\overline{s_n}\overline{t_m}) = @_{m+n}\big(\cdots @_{n+1}(@_n(\cdots @_1(x, \tau(s_1))\cdots, \tau(s_n)), \tau(t_1))\cdots, \tau(t_m)\big)$.
Bei k definierten Funktionssymbolen der Stelligkeiten $a_1, \ldots, a_k$ und k' Konstruktoren der Stelligkeiten $a_{k+1}, \ldots, a_{k+k'}$ müssen demnach im ersten Schritt (innerstes @-Symbol) $\sum_{i=1}^{k+k'} a_i$ Regeln ausprobiert werden, weil für jedes Funktionssymbol der Stelligkeit a_k genau a_k neue Regeln (mit @ als Kopfsymbol der linken Seite) zusätzlich eingeführt wurden. Für jede Anwendung einer Regel für ein Funktions- oder Konstruktorsymbol mit Index i sind damit die nächsten $a_i - 1$ NN-Schritte festgelegt (s. Abschnitt 4.4) – und können insbesondere bei einer geschickten Implementierung in konstanter Zeit "gefunden" werden –, bis nämlich zu einer vollapplizierten Funktion oder einem vollapplizierten Konstruktor ausgewertet wurde (das sind die Regeln der Form $@(\$f(x_1, \ldots, x_{j-1}), x_j) \to f(x_1, \ldots, x_j)$ für $j \in \mathbb{N}$). Nach Auswertung dieser vollapplizierten Funktion mit Resultat $\sigma(\tau(r))$ erhält man das Ergebnis
$@_{m+n-a_i}(\cdots @_1(\sigma(\tau(r)), \tau(t_{a_i+1}))\cdots, \tau(t_{m+n-a_i}))$.
Dabei kann $a_i = n$ gesetzt werden (a_i ist wie n die Stelligkeit eines Funktionssymbols), falls die angewendete Regel eine für ein Funktions- und kein Konstruktorsymbol ist, so daß das Resultat wie im [ona]-Fall dem Resultat der NCA-Ableitung entspricht. Wenn eine Regel für ein Konstruktorsymbol gewählt wurde, so führt NCAan dieser Stelle einen [dv]-Schritt durch. Offensichtlich muß NN n Schritte durchführen (Term $\overline{s_n}$), wohingegen NCA einen benötigt.

Demnach muß NN $\sum_{i=1}^{k+k'} a_i$ Regeln im ersten Schritt und NCA $\sum_{i=1}^{k} b_i \cdot a_i$ Regeln ausprobieren.
Damit ergibt sich die Notwendigkeit des Ratens von $\sum_{i=1}^{k}(b_i - 1)\cdot a_i + \sum_{i=1}^{k} a_i$ Regeln für NCA (die Stelligkeiten sind nach Vereinbarung größer als 0) und von $\sum_{i=1}^{k'} a_{k+i} + \sum_{i=1}^{k} a_i$ Regeln für NN. Es stellt sich also die Frage, wie das Verhältnis der ersten Summanden $\sum_{i=1}^{k}(b_i - 1)\cdot a_i$ und $\sum_{i=1}^{k'} a_{k+i}$ aussieht, um eine vergleichende Abschätzung des Aufwands vornehmen zu können.
Unterstellt man nun eine durchschnittliche Konstruktoren- und Funktionsstelligkeit $a_c = a_f$, und approximiert man $k = k'$, so folgt $a_c \cdot k' < a_f \cdot \sum_{i=1}^{k}(b_i - 1)$. Das Verhältnis beider Ungleichungsseiten wird sich insbesondere dann positiv für NN auswirken, wenn Datenbanken (Prädikate als boolesche Funktionen) implementiert werden, bei denen pro Funktionssymbol sehr viele Regeln existieren werden.
Vernachlässigt man dann den Aufwand, den NN für das "Finden" der nächsten

$i-1$ Regeln benötigt, so zeigt sich, daß die von NCA durchzuführenden Operationen viel umfangreicher sind, zumal der Term $x\overline{s_n}\,\overline{t_m}$ geeignet partitioniert werden muß, was aufwendige Listenoperationen beinhaltet (wenn man die applikativen Terme als Listen ablegt, müssen diese an der Stelle $n+1$ geteilt werden).

Es kann also davon ausgegegangen werden, daß das Verhalten von NN in dieser Situation "besser" als das von NCA ist.

Eine Zusammenfassung und Illustration des Komplexitätsvergleichs anhand von Beispielen wird im übernächsten Abschnitt gegeben.

Es stellt sich zuletzt natürlich die Frage, wann die als problematisch identifizierten Situationen ($m > 0$ für die [onv]- und [ona]-Inferenz) auftreten. Das ist genau dann der Fall, wenn auf rechten Regelseiten partielle Funktionsapplikationen auftreten (denn andernfalls würden Terme der Form $r\overline{t_m} \approx t$ niemals gelöst werden können). Wenn man genau diesen Fall verbietet, dann ergibt sich die Simulation der [ona]-Inferenz sofort (sie entspricht dann nämlich einer [on]-Inferenz des OINC-Kalküls). Die [onv]-Inferenz für einen Term $x\overline{s_n} \approx t$ wird durch $n+1$ [on]-Inferenzen mit den verschiedenen partiell applizierten "Instanzen" einer Regel simuliert.

Beispiel 4.5.7. *(partielle Applikation auf rechten Regelseiten)*
$\mathcal{R} = \{inc \to add_1, add_1\ x \to s\ x\}$ sei ein ATES, in dem auf einer rechten Regelseite eine partielle Funktionsapplikation auftritt (add_1).
Das entsprechende System mit expliziter Applikation und den neuen Regeln ist $\mathcal{R}^@ = \{inc \to \$add_1, add_1(x) \to s(x), @(\$add_1, x) \to add_1(x)\}$.
Dann gibt es bzgl. $\mathcal{R}_\equiv$ die NCA-Ableitung

$$
\begin{array}{rll}
inc\ x \equiv s\ 0 \approx true & \rightsquigarrow_{[ona1]} & inc\ x \approx s(x_1), s\ 0 \approx s(y_1), x_1 \equiv y_1 \approx true \\
\Longrightarrow^2 & \rightsquigarrow_{[ona2]} & add_1\ x \approx s(x_1), s\ 0 \approx s(y_1), x_1 \equiv y_1 \approx true \\
 & \rightsquigarrow_{[ona3]} & x \approx x', s(x') \approx s(x_1), s\ 0 \approx s(y_1), \\
 & & x_1 \equiv y_1 \approx true \\
\Longrightarrow^3 & \rightsquigarrow_{[v],\{x' \mapsto x\}} & s(x) \approx s(x_1), s\ 0 \approx s(y_1), x_1 \equiv y_1 \approx true \\
 & \rightsquigarrow^2_{\{x_1 \mapsto x\}} & s\ 0 \approx s(y_1), x \equiv y_1 \approx true \\
\Longrightarrow^1 & \rightsquigarrow^2_{\{y_1 \mapsto 0\}} & x \equiv 0 \approx true \\
 & \rightsquigarrow_{[ona4]} & x \approx 0, 0 \approx 0, true \approx true \\
\Longrightarrow^4 & \rightsquigarrow^2_{\{x \mapsto 0\}} & true \approx true \\
 & \rightsquigarrow_{[da]} & \Box
\end{array}
$$

und bzgl. $\mathcal{R}^@_\equiv$ die NN-Ableitung

$$\begin{array}{rll} @(inc, x) \equiv s(0) & \rightsquigarrow & @(\$add_1, x) \equiv s(0) \\ & \rightsquigarrow & add_1(x) \equiv s(0) \\ & \rightsquigarrow & s(x) \equiv s(0) \\ & \rightsquigarrow & x \equiv 0 \\ & \rightsquigarrow_{\{x \mapsto 0\}} & true \end{array}$$

Dabei entsprechen die ersten beiden NN-Schritte der zweiten [ona]-Inferenz, der dritte NN-Schritt der dritte [ona]-Inferenz und der vierte bzw. fünfte NN-Schritt der ersten bzw. vierten [ona]-Inferenz, wie man an den durch $\Longrightarrow$ gekennzeichneten Lösungszeitpunkten in der NCA-Ableitung erkennen kann.

■

Die Simulation der [ona]-Inferenz geschieht dann bei Abwesenheit von partiell applizierten Funktionen auf rechten Regelseiten in OINC durch eine [on]-Inferenz.

Lemma 4.5.8.
Wenn in einer erfolgreichen NCA-Ableitung eine [ona]-Inferenz durchgeführt wird, kann von OINC bzgl. des nicht-applikativen Systems eine [on]-Inferenz unter Anwendung der gleichen (ebenfalls nicht applikativen) Regel stattfinden.

Beweis
Auf linken Gleichungsseiten sind die Terme, auf die die [ona]-Inferenz angewendet wird, stets von der Form $f\overline{s_n}\overline{t_m}$ mit $m = 0$ (andernfalls entstünde als Resultat der Inferenz ein Term $r\overline{t_m}$, und da auf rechten Seiten keine partiell applizierten Funktionen auftauchen sollen, könnte im Fall $m > 0$ niemals eine Lösung existieren). Die NCA-Inferenz ist demnach von der Form $f\overline{s_n} \approx t \rightsquigarrow_{[ona]} s_1 \approx l_1, \dots, s_n \approx l_n, r \approx t$ bei Verwendung einer frischen Regel $f\overline{l_n} \to r$. Im Fall $n < Arity(f)$ wird eine [da]-Inferenz durchgeführt (Lemma 4.5.2).
Die OINC-Inferenz bzgl. des (nicht applikativen) Systems hat wegen $n = Arity(f)$ dann das identische Aussehen.

□

Für die [onv]-Inferenzen ergibt sich ein Zusammenhang, der der Simulation der [dv]-Inferenz ähnelt:

Lemma 4.5.9.
Wenn es bzgl. eines Systems $\mathcal{R}$ ohne partiell applizierten Funktionen auf rechten Regelseiten in einer erfolgreichen Ableitung eine NCA-Inferenz

$$x\overline{s_n}\overline{t_m} \approx t, E \rightsquigarrow_{[onv]} \vartheta(s_1 \approx l_1, \dots, s_n \approx l_n, r\overline{t_m} \approx t, E)$$

mit $\vartheta = \{x \mapsto f\overline{u_k}\}$ für eine frische Regel $f\overline{u_k}\overline{l_n} \to r$ gibt, dann

1. *ist $m = 0$ und*

2. *deshalb gibt es bzgl. $\mathcal{R}^{@}$ (in nicht-applikativer Notation) eine OINC-Ableitung*

$$\begin{aligned}\tau(x\overline{s_n}) \approx \tau(t), \tau(E) \quad \rightsquigarrow^* \quad & \vartheta'\big(\tau(s_1) \approx \tau(l_1), \ldots, \tau(s_n) \approx \tau(l_n),\\ & \tau(r) \approx \tau(t), \tau(E)\big)\end{aligned}$$

mit $\vartheta' = \{x \mapsto \tau(f\overline{u_k})\}$.

Beweis
$m = 0$ folgt aus der Tatsache, daß auf rechten Regelseiten keine partiell applizierten Funktionssymbole zugelassen sind.

Dann ist $\tau(x\overline{s_n}) = @_n(\cdots(@_1(x, \tau(s_1)), \tau(s_2))\cdots, \tau(s_n))$, und es gibt ähnlich wie im Fall der Simulation der [dv]-Inferenz die OINC-Ableitung

$$\begin{aligned}
& @_n(\cdots @_1(x, \tau(s_1))\cdots, \tau(s_n)) \approx \tau(t), \tau(E)\\
\rightsquigarrow_{[on]}\quad & @_{n-1}(\cdots @_1(x, \tau(s_1))\cdots, \tau(s_{n-1})) \approx \$f(x_1^{(1)}, \ldots, x_{k+n-1}^{(1)}),\\
& \tau(s_n) \approx x_{k+n}^{(1)}, f(x_1^{(1)}, \ldots, x_{k+n}^{(1)}) \approx \tau(t), \tau(E)\\
\rightsquigarrow_{[on]}\quad & @_{n-2}(\cdots @_1(x, \tau(s_1))\cdots, \tau(s_{n-2})) \approx \$f(x_1^{(2)}, \ldots, x_{k+n-2}^{(2)}),\\
& \tau(s_{n-1}) \approx x_{k+n-1}^{(2)}, \$f(x_1^{(2)}, \ldots, x_{k+n-1}^{(2)}) \approx \$f(x_1^{(1)}, \ldots, x_{k+n-1}^{(1)}),\\
& \tau(s_n) \approx x_{k+n}^{(1)}, f(x_1^{(1)}, \ldots, x_{k+n}^{(1)}) \approx \tau(t), \tau(E)\\
\rightsquigarrow_{[on]}^{n-2}\quad & x \approx \$f(x_1^{(n)}, \ldots, x_k^{(n)}),\\
& \tau(s_1) \approx x_{k+1}^{(n)}, \$f(x_1^{(n)}, \ldots, x_{k+1}^{(n)}) \approx \$f(x_1^{(n-1)}, \ldots, x_{k+1}^{(n-1)}),\\
& \tau(s_2) \approx x_{k+2}^{(n-1)}, \$f(x_1^{(n-1)}, \ldots, x_{k+2}^{(n-1)}) \approx \$f(x_1^{(n-2)}, \ldots, x_{k+2}^{(n-2)}),\\
& \quad\vdots\\
& \tau(s_{n-1}) \approx x_{k+n-1}^{(2)}, \$f(x_1^{(2)}, \ldots, x_{k+n-1}^{(2)}) \approx \$f(x_1^{(1)}, \ldots, x_{k+n-1}^{(1)}),\\
& \tau(s_n) \approx x_{k+n}^{(1)}, f(x_1^{(1)}, \ldots, x_{k+n}^{(1)}) \approx \tau(t), \tau(E)\\
\rightsquigarrow_{[d,v]}^{+}\quad & f(x_1^{(n)}, \ldots, x_k^{(n)}, \tau(s_1), \ldots, \tau(s_n)) \approx \tau(t), \tau(E)\\
\rightsquigarrow_{[on]}\quad & x_1^{(n)} \approx \tau(u_1), \ldots, x_k^{(n)} \approx \tau(u_k), \tau(s_1) \approx \tau(l_1), \ldots, \tau(s_n) \approx \tau(l_n),\\
& \tau(r) \approx \tau(t), \tau(E)\\
\rightsquigarrow_{[v]}^{k}\quad & \sigma\big(\tau(s_1) \approx \tau(l_1), \ldots, \tau(s_n) \approx \tau(l_n), \tau(r) \approx \tau(t), \tau(E)\big)
\end{aligned}$$

unter Anwendung der Regel $\tau(f\overline{u_k s_n} \to r)$ im letzten [on]-Schritt und der Substitution $\sigma = \{x_1^{(n)} \mapsto \tau(u_1), \ldots, x_k^{(n)} \mapsto \tau(u_k)\}$.
Die insgesamt errechnete Substitution ist damit wegen der Bindung von x an $\$f(x_1^{(n)}, \ldots, x_k^{(n)})$ das Resultat $\{x \mapsto \$f(\tau(u_1), \ldots, \tau(u_k))\}$. Man vergegenwärtige sich erneut, daß außer der ersten und der letzten die [on]-Inferenzen deterministisch festgelegt sind.

□

Damit ist nun die Simulation von NCA durch OINC für Systeme gezeigt, die partiell applizierte Funktionen auf rechten Regelseiten zulassen. Aus der wechselseitigen Simulation von OINC und NN folgt damit für diesen Fall auch die Simulation durch NN.

Satz 4.5.10.
Bei Verwendung von TES, die keine partiell applizierten Funktionen auf rechten Regelseiten besitzen, kann jeder NCA-Ableitung eindeutig eine OINC-Ableitung mit der gleichen Lösung (modulo der Transformation τ und Variablenumbenennung) zugeordnet werden. Damit korrepsondiert auch mit jeder erfolgreichen NCA-Ableitung eine NN-Ableitung mit derselben Lösung.

Beweis
Folgt aus den Lemmata 4.5.2, 4.5.4, 4.5.6, 4.5.8 und 4.5.9. Die Übereinstimmung mit NN folgt dann aus Satz 3.3.35.

□

4.6 Strikte Gleichheit: s-NCA

NCA wird in [NMI95] auf eine ähnliche Weise wie OINC zu einem Kalkül s-NCA erweitert, indem Regeln für die strikte Gleichheit in den Kalkül "hineincodiert" werden. Eine Analyse der Inferenzregeln läßt vermuten, daß sich wie im Fall von s-OINC eine Simulation zeigen läßt, wenn man die Ableitung so unterteilt, daß Teilableitungen bis zu einem Narrowingschritt an einer Position mit $\equiv$ untersucht werden.
Das soll hier nicht geschehen, weil die Simulation von NCA durch NN mit der Warren'schen Methode nicht formal gezeigt wurde.

Es ist aber davon auszugehen, daß die Simulation von NN durch s-NCA in derselben Weise geschehen kann, wie das für die Simulation von NN durch s-OINC geschehen ist.

4.7 Vergleich der Komplexitäten

Wenn keine Funktionen höherer Ordnung vorhanden sind, dann verhält sich NCA wie OINC für nicht-applikative Systeme (das folgt sofort aus einer direkten Gegenüberstellung der Regeln; es entstehen insbesondere keine "kritischen" Gleichungsseiten der Form $r\overline{t_m}$ für eine rechte Gleichungsseite). Wie im letzten Kapitel gezeigt wurde, ist das Verhalten von OINC und damit das von NCA sehr viel schlechter als das von NN.

Beispiel 4.7.1. *(NCA vs. OINC im Fall erster Ordnung)*
Es sei $\mathcal{S}$ das TES aus Beispiel 2.1.6 mit den Regeln für die Addition und die Standardordnung auf natürlichen Zahlen in "applikativer Schreibweise", d.h. die Terme werden als applikative Terme aufgefaßt.
Dann gibt es für die Anfrage $y \leq y + z \approx true$ die NCA-Ableitung (applikative Terme, $\leq$ und $+$ präfix notiert)

$$
\begin{array}{llll}
\leq\ x\ (+\ y\ z) \approx true & \leadsto_{[ona]} & x \approx 0, +\ y\ z \approx x', true \approx true \\
& \leadsto_{[dv],\{x \mapsto 0\}} & +\ y\ z \approx x', true \approx true \\
& \leadsto_{[v],\{x' \mapsto +\ y\ z\}} & true \approx true \\
& \leadsto_{[da]} & \Box
\end{array}
$$

und die OINC-Ableitung bzgl. des Systems in nicht-applikativer Sschreibweise

$$
\begin{array}{lll}
x \leq y + z \approx true & \leadsto_{[on]} & x \approx 0, y + z \approx x', true \approx true \\
& \leadsto_{[v],\{x \mapsto 0\}} & y + z \approx x', true \approx true \\
& \leadsto_{[v],\{x' \mapsto y+z\}} & true \approx true \\
& \leadsto_{[d]} & \Box
\end{array}
$$

In beiden Ableitungen werden sich entsprechende Inferenzen angewendet ([ona]–[on], [dv]–[v], [da]–[d]).

■

Damit ergibt sich auch sofort – selbst wenn das Verhalten von NCA bei Programmen mit Funktionen höherer Ordnung nicht untersucht würde –, daß NN im Normalfall NCA vorzuziehen sein wird, weil Konstrukte höherer Ordnung sicherlich seltener als solche erster Ordnung auftreten.

Bei Vorhandensein solcher Konstrukte wurden folgende Eigenschaften ermittelt:

1. Für einen [dv]-Schritt ausgehend von $x\overline{s_n} \approx f\overline{t_m}\overline{u_n}$ muß OINC bzgl. $\mathcal{R}^{@}$ n [on]-, $n+1$ [d]- und $(n^2+3n+2)/2$ [v]-Inferenzen durchgeführt werden. Dabei müssen bei k definierten Funktionen, für die jeweils b_k Regeln existieren, $n \cdot \sum_{i=1}^{k} b_i$ Regeln bei Anwendung der [on]-Schritte geraten werden; eine sehr teure Operation. NN hingegen wertet zunächst den zweitinnersten Term $@_1(x, \tau(s_1))$ aus, was bei k Regeln für Funktionen der Stelligkeit $a_1, \ldots, a_k$ das Raten von $\sum_{i=1}^{k} a_i$ Regeln im ersten Schritt erfordert. Diese legen dann allerdings die jeweils nächsten $a_k - 1$ anzuwendenden Regeln fest. Insgesamt ist davon auszugehen, daß NCA in diesem Fall ein besseres rechnerisches Verhalten als NN an den Tag legt.

2. Für die [ona]-Inferenzen wurde bereits im letzten Kapitel festgestellt, daß das Verhalten von NCA i.a. schlechter als das von NN sein wird, weil NCA wie OINC die Regeln zu früh rät und überdies Terme mit n Argumenten in alle möglichen Partitionierungen von n in zwei Summanden aufspalten muß.

3. Für die [onv]-Inferenzen wurde im letzten Kapitel ausführlich diskutiert, daß das Verhalten NN "besser" sein wird (teure "Partitionierung" der linken Gleichungsseiten).

4. Da die Auswertung von Termen mit Variablen als Kopfsymbol im Vergleich zur Auswertung von Termen mit Konstanten als Kopfsymbol eher die Ausnahme sein wird, ist davon auszugehen, daß das Verhalten von NN i.a. besser als das von NCA ist (s.o.; NCA verhält sich wie OINC

in nicht-applikativen Systemen). Für die Auswertung von Konstrukten höherer Ordnung kann eine solche Aussage nicht getroffen werden; wahrscheinlich ist das Verhalten in etwa gleich gut ("Vorteile" für NCA bei [dv], Nachteile bei [ona] und [onv]).

Zur Illustration sollen die folgenden Benchmarks dienen. Dabei kommen die TES aus den Beispielen 4.7.2 und 4.7.3 zum Einsatz (weitere Beispiele findet man in [Pre95a] oder [Pre95b]).

Beispiel 4.7.2. *([GHR92], [NMI95])*
$\mathcal{R}$ sei ein TES mit den Regeln

$$\mathcal{R} = \left\{ \begin{array}{rcl} plus(0, X) & \rightarrow & x \\ plus(s(X), Y) & \rightarrow & s(plus(X, Y)) \\ double(X, X) & \rightarrow & plus(X, X) \\ map(F, []) & \rightarrow & [] \\ map(F, [X|Y]) & \rightarrow & [@(F, X)|map(F, Y)] \\ compose(F, G, X) & \rightarrow & @(F, @(G, X)) \end{array} \right\}$$

Dann kann bzgl. $\mathcal{R}^{@}_{\equiv}$ die Anfrage

$$map(F, [s(0), 0]) \equiv [s(s(s(0))), s(0)]$$

gestelt werden, die u.a. die Lösung $F = \$compose(\$s, \$double)$ besitzt, d.h. $F \stackrel{\wedge}{=} \lambda x.compose\big(\lambda y.s(y), \lambda z.double(z), x\big)$. Man mache sich klar, daß s eigentlich keine Funktion, sondern ein Konstruktor ist.

■

Im folgenden bezeichne $s^n(0)$ den Term $\overbrace{s(s \cdots s}^{n-mal}(0) \cdots)$.
Für die erste Tabelle ist das Ziel $G_m =$
$map(F, [s^m(0), s^{m-1}(0), \ldots, s(0), 0]) \equiv [s^{2m+1}(0), s^{2m-1}(0), \ldots, s^3(0), s(0)]$
für verschiedene Werte von m verwendet worden.

Die Implementierung von NCA läuft bei Lösung der Anfrage $G_m \approx true$ in eine Endlosschleife. Das liegt an der Unvollständigkeit der in PROLOG implementierten Resolution als Tiefensuche: F wird an Terme der Form $compose\ (plus\ 0) compose(plus\ 0) \cdots$ gebunden; ein Umstellen der Regeln kann das nicht verhindern. Der Grund liegt darin, daß $\equiv$ über *map* steht und die Regeln für die strikte Gleichheit geordnet sind. Zunächst versucht NCA bei einer entsprechenden Sortierung der Regeln nämlich, die linke strikte Gleichungsseite an eine (generische) Liste zu binden, was über eine [ona]-Inferenz geschieht. Es folgen einige Schritte (z.B. mit der Bindung von F an *double*), woraufhin die strikte Gleichheit von $double(s^m(0))$ und $s^{2m+1}(0)$ getestet wird, was natürlich fehlschlägt. Nach einigen weiteren Schritten wird dann F an die (noch nicht) applizierte Funktion *compose* gebunden, und der Vorgang wiederholt sich (mit der partiellen Applikation von *compose* auf den genannten unendlichen Term). Eine Umstellung der Regeln für die strikte Gleichheit löst das Problem nicht; vielmehr würde dann gleich im ersten Schritt versucht, die linke Seite modulo den Regeln mit z.B. 0 zu unifizieren, was fehlschlägt. Dann wird irgendwann F an *compose* gebunden, und es tritt das gleiche Problem wie im anderen Fall auf.

Eine Implementierung von NCA mit Breitensuche (durch iterierte Tiefensuche) ist in bezug auf die Effizienz inakzeptabel; für G_1 wurde die Berechnung nach acht Stunden ergebnislos abgebrochen.

Deshalb wird im folgenden für NCA die Anfrage nicht mit strikter, sondern mit reflexiver Gleichheit gestellt. Die gemessenen Werte sind dann aus zwei Gründen mit Vorsicht zu vergleichen: Zum einen ist die Implementierung von NN mirt Warren's Methode wiederum compiliert, und zum anderen wird für NCA die Anfrage ohne strikte Gleichheit gestellt. Die Regeln für die strikte Gleichheit wurden nicht in die Implementierung aufgenommen. Die Implementierung von NN mit Warren's Methode ist nicht optimiert.

Die Berechnungen wurden nach Ermittlung der ersten Lösung $compose(s, double)$ gestoppt. Tabelle 7 gibt einen Überblick; Tabelle 8 schlüsselt die Zahlen für NCA auf. Dabei gelten dieselben Bezeichnungen für Prädikate wie in der Implementierung von OINC.

	NN		NCA		Verhältnis	
m	CP	Time	CP	Time	CP	Time
1	26	53	246	447	1:9	1:8
2	34	80	379	708	1:11	1:9
3	43	114	532	1005	1:12	1:9
4	52	157	701	1334	1:13	1:8
5	61	203	886	1694	1:15	1:8
10	106	524	2051	3947	1:19	1:8
20	196	1570	5581	10717	1:28	1:7
50	466	7961	25771	49146	1:55	1:6
100	916	29448	91421	173599	1:100	1:6

Tab. 7: Vergleich der Anzahl der Choice-Points bei Programmen höherer Ordnung

	solve		getRule		createEq	
m	CP	Time	CP	Time	Calls	Time
1	215	378	31	23	243	46
2	328	593	51	41	389	74
3	461	841	71	57	564	107
4	609	1113	92	76	763	145
5	772	1412	114	95	986	187
10	1812	3283	239	194	2461	470
20	5017	8905	564	435	7211	1377
50	23632	40825	2139	1476	35861	6845
100	84657	144210	6764	4274	131611	25115

Tab.8: Profile der Berechnungen aus Tab. 7

Beispiel 4.7.3. *(vgl. Kap. 2.1. in [KA96])*
Es sei $\mathcal{R}$ ein TES mit den Regeln

$$\left\{\begin{array}{l} oneHom(E_1,E_2,Phi,O_G,O_H) \rightarrow \\ \quad @\big(Phi, @(@(O_G,E_1),E_2)\big) \equiv @\big(@(O_H,@(Phi,E_1)),@(Phi,E_2)\big), \\ hom([\_|\_],[],\_,\_,\_) \rightarrow true, hom([],[\_|\_],\_,\_,\_) \rightarrow true, \\ hom([G_1|GR_1],[G_2|GR_2],Phi,O_G,O_H) \rightarrow \\ \quad oneHom(G_1,G_2,Phi,O_G,O_H) \\ \quad \wedge hom([G_1],GR_2,Phi,O_G,O_H) \wedge hom(GR_1,[G_2|GR_2],Phi,O_G,O_H), \\ mod_2(0) \rightarrow 0, mod_2(s(0)) \rightarrow s(0), mod_2(s(s(X))) \rightarrow mod_2(X), \\ mod_4(0) \rightarrow 0, mod_4(s(0)) \rightarrow s(0), mod_4(s(s(0))) \rightarrow s(s(0)), \\ mod_4(s(s(s(0)))) \rightarrow s(s(s(0))), mod_4(s(s(s(s(X))))) \rightarrow mod_4(X), \\ add_2(X,Y) \rightarrow mod_2(plus(X,Y)), \\ add_4(X,Y) \rightarrow mod_4(plus(X,Y)), \\ plus(0,X) \rightarrow X, plus(s(X),Y) \rightarrow s(plus(X,Y)) \end{array}\right\}$$

unter Verwendung von _ für anonyme Variablen.
Die Funktion $hom(G,H,Phi,O_G,O_H)$ testet, ob Phi ein Homomorphismus zwischen der Gruppe $\langle G,O_G\rangle$ und der Gruppe $\langle H,O_H\rangle$ (unter Vernachlässigung des neutralen Elements) ist. add_2 und add_4 implementieren die Addition modulo 2 bzw. 4.
Dann kann bzgl. $R_{\equiv}^{@}$ die Anfrage

$$hom([0,s(0),s(s(0)),s(s(s(0)))],[0,s(0)],F,\$add_4,\$add_2) \equiv true$$

gestellt werden, deren Resultat u.a. $F = \$mod_2$, d.h. $F \triangleq \lambda x.mod_2\ x$, ist.

■

In diesem Beispiel kann die Anfrage für NCA wiederum nicht als
$hom([0,s(0),s(s(0)),s(s(s(0)))],[0,s(0)],F,\$add_4,\$add_2) \equiv true \approx true$

gestellt werden, weil sich dieselben Probleme wie im letzten Beispiel ergeben. Allerdings müssen für dieses Beispiel die Regeln für $\equiv$ in die Implementierung aufgenommen werden, weil $\equiv$ auf einer rechten Gleichungsseite auftritt.
Das experimentelle Resultat ist in Tabelle 9 und Tabelle 10 abgebildet.

NN		NCA		Verhältnis	
CP	Time	CP	Time	CP	Time
54	166	7862	12559	1:146	1:76

Tab.9: Vergleich der Anzahl der Choice-Points bei Berechnung eines Homomorphismus

`solve`		`getRule`		`createEq`	
CP	Time	CP	Time	Calls	Time
6990	10454	872	510	8238	1595

Tab.10: Profil der Berechnung aus Tab. 9

Wiederum schneidet NN mit Warren's Methode deutlich "besser" ab. Gründe dafür sind:

1. Der Nichtdeterminismus tritt in NCA nach wie vor "zu früh" auf. Das führt insbesondere dazu, daß bei einer (effizienten) Tiefensuchstrategie bei bestimmten Anfragen $\equiv$ als Kopfsymbol ausscheidet (s.o.).

2. Die Codierung applikativer Terme ist aufwendig. *Alle* Terme müssen codiert werden; ein Vorteil von NN liegt darin, daß nur Terme mit einer Variablen als Kopfsymbol in die applikative Schreibweise übersetzt werden müssen. Daraus folgt sofort ein Nachteil von NCA bei der Behandlung von Programmen und Anfragen erster Ordnung.

Die bisherigen Resultate müssen immer auch unter dem Gesichtspunkt betrachtet werden, daß die zugrundeliegenden TES eine unterschiedliche "Mächtigkeit" besitzen (induktiv-sequentiell vs. orthogonal). Darauf wird u.a. im nächsten Abschnitt eingegangen.

4.8 Diskussion

Diskutiert werden die Ergebnisse dieses Kapitels, die Problematik unendlicher Berechnungen, die Verwendung von Typinformationen zur Reduktion des Suchraums, die Ausdrucksstärke der zugrundeliegenden TES (induktiv-sequentiell vs. orthogonal) und alternative Ansätze mit Unifikation höherer Ordnung.

1. Zusammenfassung und Ergebnisse
Dieses Kapitel beschäftigte sich mit zwei Ansätzen, das Konzept der Funktionen als "first class citizens" in den funktional-logischen Kontext einzubetten.
Zum einen wurde eine syntaktische Transformation beschrieben, "Warren's Methode" genannt, die neue Konstruktoren für partiell applizierte Funktionen einführt. Zum anderen wurde eine Klasse von TES definiert, die einzelne Teilterme eines Terms durch ein implizites Applikationssymbol verknüpft, die applikativen TES. Applikative TES gestatten Terme mit Variablen als Kopfsymbol.

Es folgte der Vergleich von NN mit Warren's Methode und NCA, einer effizienten Adaptation von OINC für applikative TES. Es ist gelungen, NN durch NCA zu simulieren. Entscheidend dabei war, daß sich bestimmte NN-Teilableitungen zu sog. "Multischritten" zusammenfassen lassen. Dieses Resultat kann als Beweis der "Korrektheit" der Warren'schen Methode angesehen werden.
Die Rückrichtung scheiterte daran, daß für die [ona]- und [onv]-Inferenzen von NCA keine NN-Ableitungen angegeben werden konnten, wenn auf rechten Regelseiten partiell applizierte Funktionen vorkommen dürfen. Verbietet man solche Regeln, so konnte die Simulation angegeben werden.

Die Analyse beider Kalküle führte zu der begründeten Annahme, daß NCA aus denselben Gründen wie OINC weniger effizient ist: Die Regeln werden früher als nötig geraten und erzeugen damit eine hohe Anzahl fehlschagender Berechnungen. Außerdem ist das Erzeugen neuer Gleichungen zeitaufwendig. Diese Einschätzung wurde anhand experimenteller Resultate bestätigt.

Es wurde auch festgehalten, daß sich NCA für applikative Programme und Anfragen erster Ordnung so verhält, wie sich OINC für die entsprechenden Programme und Anfragen in nicht-applikativer Schreibweise verhält. Da Konstrukte höherer Ordnung in der Praxis sicherlich weitaus seltener anzutreffen sind als solche höherer Ordnung, ergibt sich aus der "Überlegenheit" von NN gegenüber OINC insbesondere in diesem Fall die Überlegenheit von NN gegenüber NCA.

2. Unendliche Berechnungen
Ein Aspekt des Narrowing höherer Ordnung wurde bisher nur gestreift: die Problematik unendlicher Berechnungen. Es wurde erläutert, warum NCA bei einer Implementierung durch Tiefensuche in speziellen Fällen (Beispiele 4.7.2 und 4.7.3 keine Lösung findet. Die Implementierung durch Breitensuche ist aus Effizienzgründen skeptisch zu beurteilen.
Allerdings können nicht nur einzelne Berechnungen unendlich lang werden (was passieren wird, wenn Funktionen als Argumente partiell applizierte Funktionen verlangen). Es wird auch vorkommen, daß die Menge der Lösungen unendlich groß ist. Das ist z.B. dann der Fall, wenn eine Identitätsfunktion auf unendlich viele Arten dargestellt werden kann; z.B. durch unendliche Komposition von Additionen von Null.
Dieser zweite Punkt ist nicht befriedigend zu lösen. Eine Möglichkeit wäre, sich auf die die erste berechnete Lösung zu beschränken. Das könnte in der Pro-

grammiersprache Curry beispielsweise durch die Anwendung der encapsulated search mit den zur Verfügung gestellten Suchoperatoren erfolgen [Han98]. Auch dabei entsteht natürlich das Problem bei einer Implementierung durch Tiefensuche (Curry stellt Breitensuche zur Verfügung; effiziente Berechnungen sind in vielen Fällen damit kaum möglich).

3. Berücksichtigung von Typen
Der Lösungsraum kann aber stark eingeschränkt werden, wenn man getyptes Narrowing durchführt (monomorphe Programme in [AT98], polymorphe Programme im funktionalen Kontext in [BBH97]). Die Idee ist, die verschiedenen polymorphen Instanzen der (impliziten oder expliziten) Funktionsapplikation mit den zulässigen Typen zu annotieren (vgl. auch Kap. 8 in [NMI95]). Eine Erweiterung von NN um Warren's Methode mit getyptem Narrowing für (polymorphe?) Programme erfordert sicherlich noch einiges an Forschungsaufwand.

4. Ausdrucksstärke der verwendeten TES
Ein weiterer bisher vernachlässigter Punkt ist die Ausdrucksstärke der verwendeten TES. NN ist auf induktiv-sequentielle TES beschränkt, OINC auf orthogonale. In Kapitel 3, insbesondere Abschnitt3.6, wurde kurz Parallel Narrowing angesprochen. Wenn sich die Vermutung bestätigt, daß OINC und Parallel Narrowing ein identisches Verhalten auf KBO-TES aufweisen, dann stellt sich die Frage, ob nicht-konstruktorbasierte orthogonale Systeme in der Praxis häufig auftreten. Das ist vermutlich nicht der Fall (Funktionssymbole auf linken Gleichungsseiten treten zwar insbesondere im Bereich der algebraischen Spezifikation durchaus auf; das Problem liegt aber darin, daß diese in Normalform sein müssen, um die Orthogonalität des gesamten Systems nicht zu beeinflussen. Ein Beispiel ist die Definition der Stackoperationen, wenn *push* und *pop* als Funktionen angesehen werden).
Programme für die symbolische Differentiation [HP96] beispielsweise können ohne weiteres weder NCA noch NN mit Warren's Methode zur Auswertung von Anfragen zugrundegelegt werden, weil die Argumente der Differentiationsfunktion nicht applizierte Funktionen sind (wie soll eine Regel der Form $\mathit{diff}(\lambda y.y, X) \rightarrow 1$ modelliert werden?). Dieses Beispiel gibt erneut Anlaß zu der Frage, wann funktionale Terme in Normalform als Argument einer linken Regelseite Sinn machen bzw. eine elegantere Formulierung von Sachverhalten zulassen, als dies in konstruktorbasierten TES der Fall ist.

5. Alternative Ansätze mit Unifikation höherer Ordnung
Dieses Kapitel beschäftigte sich mit Verfahren, die die Unifikation höherer Ordnung durch die Verwendung von TES mit impliziter oder expliziter Funktionsapplikation vermeiden. Andere Ansätze mit Unifikation höherer Ordnung finden sich z.B. in [HP96] (Narrowing mit definierenden Bäumen höherer Ordnung), [Pre95a] (optimales Narrowing höherer Ordnung für normale konditionale TES höherer Ordnung), [Qia94] (Integration mit Narrowing erster Ordnung), [KA96], [Kuc95] (HO-Babel mit sog. Kombinatorreduktion statt β-Reduktion) oder [NM88] (λ-Prolog).

Kapitel 5

Implementierung

Dieses Kapitel erläutert die Implementierungen für NN, OINC, s-OINC und NCA. Die Implementierungen sind alle nicht für den direkten Einsatz in einer funktional-logischen Sprache gedacht. Deshalb fehlen Ein-/Ausgaberoutinen, eine Fehlerbehandlung u.ä. Insbesondere werden die resultierenden Substitutionen nicht aufbereitet, sondern müssen aus den Antworten von Prolog entnommen werden.

5.1 NN

Die Implementierung von NN für Programme erster Ordnung wurde bereits in Kapitel 3 vorgestellt. Sie basiert auf [Han95] (ausführliche Dokumentation) und erfolgte in Sicstus-Prolog 3.5.

Anfragen werden durch `?- Links===Rechts.` gestellt:

```
:-op(700,xfx,===).
```

Die Regeln für die definierten Funktionen `plus`, `leq`, `double`, `map` und `compose` wurden automatisch generiert und sind nicht optimiert:

```
plus(A,B,C):-hnf(A,D),plus_1(D,B,C).
plus_1(0,A,B):-hnf(A,B).
plus_1(s(A),B,s(plus(A,B))).
leq(A,B,C):-hnf(A,D),leq_1(D,B,C).
leq_1(0,A,true).
leq_1(s(A),B,C):-hnf(B,D),leq_1s_2(s(A),D,C).
leq_1s_2(s(A),0,false).
leq_1s_2(s(A),s(B),C):-hnf(leq(A,B),C).
double(A,B):-hnf(plus(A,A),B).
map(F,X,R):-hnf(F,HF),hnf(X,HX),map_2(HF,HX,R).
map_2(F,[],[]).
map_2(F, [X|Y], [First|map(F,HY)]):-
    hnf(X,HX), hnf(apply(F,HX),First), hnf(Y,HY).
compose(F,G,H,Result):-
```

```
   hnf(F,F1), hnf(G,G1), hnf(H,H1), hnf(apply(F1,apply(G1,H1)),Result).
```

Die Regeln für die `apply`-Funktion müssen ggf. rekursiv für das erste Argument aufgerufen werden (LR-DTs). Diese Regeln wurden nicht automatisch erzeugt. Die Reihenfolge entscheidet darüber, ob die Berechnungen terminieren oder nicht.

```
apply(A,B,C):-
   (nonvar(A), A=apply(A1,A2)) -> (
                                     apply(A1,A2,A3),
                                     apply(A3,B,C)
                                  )
                                ;    apply_1(A,B,C).
```

Für die Konstruktoren ergeben sich die Regeln

```
apply_1('$s', X, s(X)).
apply_1('$.',X,'$.'(X)).
apply_1('$.'(X),Y,[HX|HY]):-hnf(X,HX),hnf(Y,HY).
```

und für die Funktionen

```
apply_1('$plus',X,'$plus'(X)).
apply_1('$plus'(X),Y,HXY):-hnf(plus(X,Y),HXY).
apply_1('$double',X,HX):-hnf(double(X),HX).
apply_1('$compose',X,'$compose'(X)).
apply_1('$compose'(X),Y,'$compose'(X,Y)).
apply_1('$compose'(X,Y),Z,HXYZ):-hnf(compose(X,Y,Z),HXYZ).
apply_1('$map',X,'$map'(X)).
apply_1('$map'(X),Y,HXY):-hnf(map(X,Y),HXY).
```

Die Regeln für die Kopfnormalform und die strikte Gleichheit wurden bereits beschrieben.

```
hnf(A,A):-var(A),!.
hnf(leq(A,B),C):-!,leq(A,B,C).
hnf(plus(A,B),C):-!,plus(A,B,C).
hnf(double(A),B):-!,double(A,B).
hnf(map(F,X),R):-!,map(F,X,R).
hnf(apply(F,A),R):-!,apply(F,A,R).
hnf(compose(F,G,H),R):-!,compose(F,G,H,R).
hnf(one(A),B):-!,one(A,B).
hnf(A=B,C):-!,(A=B->C=true;C=false).
hnf(A,A).

A===B:-seq(A,B).
seq(A,B):-hnf(A,C),hnf(B,D),seq_1_2(C,D).
seq_1_2(0,0).
seq_1_2(false,false).
seq_1_2(s(A),s(B)):-seq(A,B).
seq_1_2([],[]).
```

```
seq_1_2([X|Xs],[Y|Ys]):-seq(X,Y),seq(Xs,Ys).
seq_1_2(true,true).
```

Für das Beispiel 4.7.3 der Homomorphismusberechnung werden nur die wesentlichen Prädikate angegeben.

```
oneHom(E1,E2,Phi,GOp,HOp,true):-
   hnf(apply(Phi,apply(apply(GOp,E1),E2)),H1),
   hnf(apply(apply(HOp,apply(Phi,E1)),apply(Phi,E2)),H2),
   seq(H1,H2).

hom(A,B,C,D,E,F):-hnf(A,G),hom_1(G,B,C,D,E,F).
hom_1([],A,B,C,D,true):-!.
hom_1(_,[],_,_,_,true):-!.
hom_1([A|B],C,D,E,F,G):-hnf(C,H),'hom_1._2'([A|B],H,D,E,F,G).
'hom_1._2'([A|B],[C|D],E,F,G,H):-
   hnf(and(oneHom(A,C,E,F,G),
       and(hom([A],D,E,F,G),hom(B,[C|D],E,F,G))),H).
```

Man beachte den Aufruf von `seq` in der Definition von `oneHom` sowie – wie im Fall von `map` vorher – die Anwendung von `apply` im Rumpf.
Die Regeln für `and` sind

```
and(A,B,C):-hnf(A,D),and_1(D,B,C).
and_1(true,A,B):-hnf(A,B).
and_1(false,A,false).
```

Stellvertretend für `mod4` und `add4` sei die Definition von `mod2` und `add2` angegeben:

```
mod2(A,B):-hnf(A,C),mod2_1(C,B).
mod2_1(0,0).
mod2_1(s(A),B):-hnf(A,C),mod2_1s_11(s(C),B).
mod2_1s_11(s(0),s(0)).
mod2_1s_11(s(s(A)),B):-hnf(mod2(A),B).

add2(A,B,C):-hnf(mod2(plus(A,B)),C).
```

Für die Benchmarks wurde auf eine Codierung der partiellen Applikation von `oneHom`, `hom` und `and` verzichtet (ein vernünftiges Vorgehen, den Suchraum einzuschränken, wenn man vorher weiß, welche Lösungen nicht von Interesse sind). Das geschieht bei NCA später auch.

```
apply_1('$mod2',X,HX):-hnf(mod2(X),HX).
apply_1('$add2',X,'$add2'(X)).
apply_1('$add2'(X),Y,HX):-hnf(add2(X,Y),HX).
```

5.2 OINC

OINC wurde als Interpreter realisiert. Eine Anfrage wird über
`?- solve([(Links,Rechts)]).` gestellt.

Die Regeln werden über `getRule` abgelegt. Das erste Argument enthält den Funktionsnamen, das zweite eine Liste der Argumente und das dritte die rechte Regelseite:

```
getRule(leq,[null,_],true).
getRule(leq,[s(_),null],false).
getRule(leq,[s(X),s(Y)], leq(X,Y)).
getRule(plus,[null,X], X).
getRule(plus,[s(X), Y], s(plus(X,Y))).
```

Das Ende der rekursiven Aufrufe von `solve`, das den Interpreter realisiert:

```
solve([]):-!.
```

Die Variableneliminationen werden in zwei Prädikaten abgelegt, um das überflüssige Anlegen von Choice-Points zu verhindern. Der Cut ist erlaubt, weil kein Nichtdeterminismus zu den anderen Inferenzregeln besteht.

```
solve([(LHS,RHS)|Equations]):-                                        %[v1]
   var(RHS),!,LHS=RHS,solve(Equations).
solve([(LHS,RHS)|Equations]):-                                        %[v2]
   var(LHS),!,LHS=RHS,solve(Equations).
```

Für die Dekompositionsregel kann die folgende Klausel als Implementierung dienen. Eine Abfrage, ob `Head` ein Funktionssymbol ist, muß nicht erfolgen, weil ein solcher Fall (gleiches Funktionskopfsymbol auf beiden Seiten) nicht bei der Verwendung induktiv-sequentieller TES auftreten kann. Der Cut ist erlaubt, weil in induktiv-sequentiellen TES kein Nichtdeterminismus mit der [on]-Inferenz auftreten kann, wenn auf rechten initialen Seiten keine Funktionssymbole auftauchen. Mit `createEquations` werden die neuen Gleichungen erzeugt.

```
solve([(LHS,RHS)|Equations]):-                                         %[d]
   LHS=..[Head|LArgs],RHS=..[Head|RArgs],
   !,                             % isTES, keine FSyms auf initialen RHS
   createEquations(LArgs,RArgs,[],NewEqs),
   append(NewEqs,Equations,NewEqs2),solve(NewEqs2).
```

Schließlich wird noch die Implementierung der [on]-Inferenz benötigt, die in dieser naiven Implementierung ebenso offensichtlich wie die anderen Inferenzregeln ist.

```
solve([(LHS,RHS)|Equations]):-                                        %[on]
   LHS=..[Head|Args],
   getRule(Head,RuleArgs,RightRuleSide),
   createEquations(Args,RuleArgs,[],NewEqs),
   append(NewEqs,[(RightRuleSide,RHS)|Equations],NewEqs2),
```

```
   solve(NewEqs2).
```

Die neuen Gleichungen werden über das folgende Prädikat erzeugt. Das erste und zweite Argument bestehen aus den linken und rechten Gleichungsseiten, das dritte ist ein Akkumulator und das vierte schließlich beinhaltet das Resultat.

```
createEquations([],[],E,E).
createEquations([L|Ls],[R|Rs],Accu,Result):-
   append(Accu,[(L,R)],NewAccu),createEquations(Ls,Rs,NewAccu,Result).
```

5.3 s-OINC

Die Implementierung von s-OINC ist der von OINC sehr ähnlich. Hier werden nur die zusätzlichen Klauseln für die zusätzlichen Inferenzregeln angegeben. Strikte Gleichungen werden durch den Funktor `strict` gekennzeichnet; damit können Anfragen mit strikter Gleichheit über
`?- solve([strict(Links,Rechts)]).` gestellt werden.
Für die symmetrischen Narrowinginferenzen für strikte Gleichheit wurde die folgende selbsterklärende Implementierung gewählt:

```
solve([strict(L,R)|Es]):-                                          %[ons1]
   nonvar(L),L=..[Head|Args],
   getRule(Head,RuleArgs,RHS),
   createEquations(Args,RuleArgs,[],NewEquations),
   append(NewEquations,[strict(RHS,R)|Es],NewEq2),
   solve(NewEq2).
solve([strict(L,R)|Es]):-                                          %[ons2]
   nonvar(R),R=..[Head|Args],
   getRule(Head,RuleArgs,RHS),
   createEquations(Args,RuleArgs,[],NewEquations),
   append(NewEquations,[strict(L,RHS)|Es],NewEq2),
   solve(NewEq2).
```

Die Dekomposition strikter Gleichungen erfolgt mit

```
solve([strict(L,R)|Es]):-                                          %[ds]
   nonvar(L),nonvar(R),
   L=..[Head|LArgs], (\+getRule(Head,_,_)),
   !,
   R=..[Head|RArgs],
   createStrictEquations(LArgs,RArgs,[],NewEquations),
   append(NewEquations,Es,NewEq2),
   solve(NewEq2).
```

Der negierte Aufruf von `getRule` garantiert dabei, daß das Kopfsymbol der Gleichungen ein Konstruktor ist. Der Cut ist erlaubt, weil kein Nichtdeterminismus mit anderen Regeln besteht. `createStrictEquations` schließlich erzeugt die

strikten Gleichungen.
Die Imitationsregeln wurden wie folgt implementiert:

```
solve([strict(L,R)|Es]):-                                         %[ims1]
   var(R),nonvar(L),
   L=..[Head|Args], (\+getRule(Head,_,_)),
   !,
   functor(L,_,N),
   length(NewVars, N),                               %neue Liste erzeugen
   R=..[Head|NewVars],
   createStrictEquations(Args, NewVars, [], NewEquations),
   append(NewEquations, Es, NewEq2),
   solve(NewEq2).
solve([strict(L,R)|Es]):-                                         %[ims2]
   var(L),nonvar(R),
   R=..[Head|Args], (\+getRule(Head,_,_)),
   !,
   functor(R,_,N),
   length(NewVars, N),                               %neue Liste erzeugen
   L=..[Head|NewVars],
   createStrictEquations(Args, NewVars, [], NewEquations),
   append(NewEquations, Es, NewEq2),
   solve(NewEq2).
```

Wiederum sorgt der negierte Aufruf von `getRule` dafür, daß die Kopfsymbole beider Gleichungsseiten Konstruktoren sind. Der Aufruf von `length` garantiert jeweils das Anlegen einer Liste mit `N` frischen Variablen, die in der Inferenzregel benötigt werden. Damit werden dann über `createStrictEquations` die neuen strikten Gleichungen erzeugt.

Die Implementierung der Elimination strikter Variablengleichungen ist offensichtlich:

```
solve([strict(L,R)|Es]):-                                           %[ts]
   var(L),var(R),L=R,solve(Es).
```

Die strikten Gleichungen werden schließlich genau wie die neuen nicht-strikten Gleichungen erzeugt:

```
createStrictEquations([],[],E,E).
createStrictEquations([L|Ls],[R|Rs],Accu,Result):-
   append(Accu, [strict(L,R)], NewAccu),
   createStrictEquations(Ls, Rs, NewAccu, Result).
```

5.4 NCA

Die Implementierung von NCA ist etwas interessanter, weil sich die Frage nach einer geeigneten Repräsentation applikativer Terme stellt. Es wurde der Ansatz gewählt, die implizite, infix notierte Applikation durch einen expliziten, präfix notierten Funktor `a` zu modellieren.
Die Umwandlung von Termen in die explizit applikative Schreibweise (d.h. unter Verwendung des Funktors `a`) erfolgt über das Prädikat `applyifyTerm`. Die Verwendung der Applikation für Terme mit einer Variablen als Kopfsymbol kann unter Angabe von `apply` (präfix notiert) erfolgen. Die Implementierung ergibt sich sofort als

```
applyifyTerm(X,X):-(atomic(X);var(X)),!.
applyifyTerm(Term,AppTerm):-
   Term=..[Head|Args],
   if(
        Head=='apply',               % explizites apply, zwei Argumente
        (
           Args=[First,VarArgs],
           applyifyTerm(First,AppFirst),
           applyifyTerm(VarArgs, AppVarArgs),
           AppTerm=a(AppFirst,AppVarArgs)
        ),
        if(
             Args==[],
             AppTerm=Head,
             (
               Args=[A1|As],
               applyifyTerm(A1, AppA1),
               appTermArgs(As,a(Head,AppA1),AppTerm)
             )
          )
     ).

appTermArgs([],Innermost,Innermost).
appTermArgs([A|As],Innermost,Result):-
   applyifyTerm(A,AppA),
   appTermArgs(As,a(Innermost,AppA),Result).
```

Anfragen können wie in der Implementation von OINC gestellt werden. Das Ende der rekursiven Aufrufs des Interpreters `solve` ist offensichtlich

```
solve([]).
```

Für die [ona]-Inferenz ergibt sich die Realisierung durch

```
solve([(Left,T)|Rest]):-                                            %[ona]
   nonvar(T),
   getHeadAndArgs(Left,F,[],Args),
   nonvar(F),
```

```
  getRule(F,RuleArgs,RHS,N),
  getFirst(Args,N,Sn,Tm),
  createEquations(Sn,RuleArgs,[],Eqs1),
  if(                                       %keine Cuts mit ..->..;..!
       Tm==[],
       NewR=RHS,
       (
         applyify(Tm,TmTerm),
         NewR=a(RHS,TmTerm)
       )
    ),
  append(Eqs1,[(NewR,T)|Rest],Eqs2),
  solve(Eqs2).
```

Dabei berechnet `getHeadAndArgs` das innermost-Symbol und die Argumente eines applikativen Ternms; der dritte Parameter ist ein Akkumulator. `getFirst(Liste,Anzahl,Erste,Letzte)` ermittelt die ersten `Anzahl` Elemente von `Liste` und speichert diese in `Erste` und den Rest von `Liste` in `Letzte`. Da keine Cuts verwendet werden dürfen (die verschiedenen Regeln müssen ausprobiert werden), wird auf `if` anstelle von `..->..;..` zurückgegriffen (Letzteres wird mit einem Cut realisiert). `applyify` erzeugt einen neuen applikativen Term mit der rechten Regelseite an der innersten und den darüberliegenden Positionen und dem Restterm desjenigen Terms, in dem das Narrowing durchgeführt wird. `getRule` erhält als weiteren Parameter die Stelligkeit der Funktion, um diese nicht jedesmal wieder neu abfragen zu müssen.

Die Dekomposition von Termen mit Variablen als Kopfsymbol wird durch

```
solve([(Left,Right)|Rest]):-                                        %[dv]
  nonvar(Right),
  getHeadAndArgs(Left,X,[],Sn),
  var(X),
  getHeadAndArgs(Right,F,[],TmUn),          %Var kann nicht auftreten!
  nonvar(F),
  length(Sn,N),
  splitRight(TmUn,N,Tm,Un),
  createEquations(Sn,Un,[],Eqs1),
  append(Eqs1,Rest,Eqs2),
  applyify([F|Tm],X),                  % mindestens Laenge 1 garantiert
  solve(Eqs2).
```

realisiert. `Right` kann wegen der Eigenschaft applikativer TES, nur Muster als linke Regelseiten zuzulassen, keine Variable sein.
`splitRight(Liste,Anzahl,Links,Rechts)` verhält sich wie `getFirst`, nur daß hier die `Anzahl` rechten Elemente von `Liste` in `Rechts` und die Restliste in `Links` gespeichert wird. `applyfiy` erzeugt wie im [ona]-Fall den neuen applikativen Term; `applyify` benötigt als erstes Argument eine Liste mit mindestens einem Argument, was garantiert wird.

Die [onv]-Inferenz ergibt sich entsprechend:

```
solve([(Left,T)|Rest]):-                                        %[onv]
   nonvar(T),
   getHeadAndArgs(Left,X,[],Args),
   var(X),
   getRule(F,RuleArgs,RHS,_),
   % F\==seq, F\=and,                      %FSyms, die nicht HO sind
   append(Uk,Vn,RuleArgs),
   length(Vn,N),
   length([S1|Ss],N),                           %neue Liste erzeugen
   append([S1|Ss],Tm,Args),
   createEquations([S1|Ss],Vn,[],Eqs1),
   applyify([RHS|Tm], NewR),               % Mind.laenge 1 garantiert
   append(Eqs1,[(NewR,T)|Rest], Eqs2),
   if(                                     %keine Cuts mit ..->..;..!
        Uk==[],
        X=F,
        (
          applyify([F|Uk], UkTerm),
          X=UkTerm
        )
     ),
   solve(Eqs2).
```

Dabei kann es wie im Fall von NN im Beispiel der Berechnung von Homomorphismen sinnvoll sein, nicht alle Funktionssymbole für die Instantiierung von Funktionsvariablen zuzulassen. Wiederum muß `if` anstelle von `..->..;..` verwendet werden, um alle Regeln ausprobieren zu können.
Für die Dekomposition applikativer Terme ergibt sich sofort die folgende Implementierung:

```
solve([(Left,Right)|Rest]):-                                     %[da]
   nonvar(Right),
   getHeadAndArgs(Left,F,[],Sn),
   nonvar(F),
   getHeadAndArgs(Right,F,[],Tn),
   createEquations(Sn,Tn,[],Eqs1),
   append(Eqs1,Rest,Eqs2),
   solve(Eqs2).
```

Diese bedarf nach dem Vorherigen keiner Erläuterung mehr, genau wie die Implementierung der Variableneliminationsregel:

```
solve([(Left,Right)|Rest]):-                                      %[v]
   var(Right),
   Right=Left,
   solve(Rest).
```

In der Implementierung wurden neben `createEquations` die folgenden

Hilfsprädikate verwendet:

```
getFirst(List,N,FirstN,LastM):-
   N>0,
   length(FirstN,N),                          %neue Liste erzeugen
   append(FirstN,LastM,List),
   !.
```

`getFirst(Liste,Anzahl,Erste,Letzte)` teilt eine `Liste` an der Position `Anzahl`, wobei die ersten Elemente in `Erste` und die letzen in `Letzte` abgespeichert werden. Dual dazu ist das Prädikat `splitRight` mit der Implementierung

```
splitRight(List,N,FirstM,LastN):-
   length(LastN,N),                           %neue Liste erzeugen
   append(FirstM,LastN,List),
   !.
```

Benötigt wurde auch `getHeadAndArgs`, ein Prädikat, das das innermost Symbol eines applikativen Terms und seine Argumente berechnet:

```
getHeadAndArgs(X,X,Accu,Accu):-
   (atomic(X);var(X)),
   !.
getHeadAndArgs(a(X,Y),Head,Accu,Args):-
   getHeadAndArgs(X,Head,[Y|Accu],Args).
```

Schließlich fand das Prädikat `applyify` Verwendung, das einen neuen applikativen Term aus einer Liste von innermost Symbol und Argumenten erzeugt.

```
applyify([A1|[]], A1):-!.        % leere Listen werden nicht angefordert
applyify([A1,A2|As], Term):-applyify([a(A1,A2)|As],Term).
```

Die Regeln für die Funktionen aus Beispiel 4.7.2 werden wieder über `getRule` abgelegt, diesmal mit der Stelligkeit der Funktion als viertem Argument. Hier finden sich auch Beispiele für die Codierung applikativer Terme.

```
getRule(plus,[0,X],X,2).
getRule(plus,[a(s,X),Y],a(s,a(a(plus,X),Y)),2).
getRule(double,[X],a(a(plus,X),X),1).
getRule(map,[_,[]],[],2).
getRule(map,[F,a(a('.',X),Y)],a(a('.',a(F,X)),a(a(map,F),Y)),2).
getRule(compose, [F,G,X], a(F,a(G,X)), 3).
```

Für die Regeln aus Beispiel 4.7.3 ergibt sich schließlich die folgende Implementierung (`mod2` und `add2` erneut stellvertretend für `mod4` und `add4`):

```
getRule(mod2,[0],0,1).
getRule(mod2,[a(s,0)],a(s,0),1).
getRule(mod2,[a(s,a(s,X))],a(mod2,X),1).
getRule(plus,[0,X],X,2).
```

In diesem Beispiel werden die Regeln für die strikte Gleichheit benötigt. Das letzte Prädikat implementiert sie für Listen.

```
getRule(and,[true,X],X,2).
getRule(and,[false,_],false,2).
getRule(seq,[0,0],true,2).
getRule(seq,[true,true],true,2).
getRule(seq,[false,false],true,2).
getRule(seq,[[],[]],true,2).
getRule(seq,[a(s,X),a(s,Y)],a(a(seq,X),Y),2).
getRule(seq,[a(a('.',X),Xs),a(a('.',Y),Ys)],
       a(a(and,a(a(seq,X),Y)),a(a(seq,Xs),Ys)),2).
```

Die Additionsprädikate ergeben sich direkt. Eine Regel für `plus` wurde bereits weiter oben angegeben; es bietet sich im Fall höherer Ordnung an, "einfache" Prädikate (d.h. Regeln, die beispielsweise nicht rekursiv sind) an den Anfang der Regeln zu stellen.

```
getRule(add2,[X,Y],a(mod2,a(a(plus,X),Y)),2).
getRule(plus,[a(s,X),Y],a(s,a(a(plus,X),Y)),2).
```

Zuletzt wird hier noch die Implementierung für die Berechnung der Homomorphismen angegeben:

```
getRule(oneHom,[E1,E2,Phi,Og,Oh],
       a(a(seq,a(Phi,a(a(Og,E1),E2))),a(a(Oh,a(Phi,E1)),a(Phi,E2))),5).
getRule(hom,[a(a('.',_),_),[],_,_,_],true,5).
getRule(hom,[[],a(a('.',_),_),_,_,_],true,5).
getRule(hom,[a(a('.',G1),GR1),a(a('.',G2),GR2),Phi,Og,Oh],
       a(a(and,a(a(a(a(a(oneHom,G1),G2),Phi),Og),Oh)),
         a(a(and,a(a(a(a(a(hom,a(a('.',G1),[])),GR2),Phi),Og),Oh)),
           a(a(a(a(a(hom,GR1),a(a('.',G2),GR2)),Phi),Og),Oh))),
       5).
```

Die rechten Regelseiten wurden in allen Fällen mit `applyifyTerm` konstruiert (`apply` als explizite Funktionsapplikation, wenn der applikative Term eine Variable als Kopfsymbol besitzt).

Auf die Angabe der trivialen Prädikate für die Übersetzung von Termen in die $s/0$-Schreibweise sowie des naiven Metainterpreters für Breitensuche wird verzichtet.

5.5 Diskussion

Die mit diesen Programmen ermittelten experimentellen Resultate für NN und OINC bzw. NCA sind nicht direkt vergleichbar, weil die Regeln im Fall von NN *compiliert* und im Fall von OINC bzw. NCA *interpretiert* werden.
Die Implementierung von NN ist nicht optimiert (geschickte Indizierung von

Argumenten durch deren Umstellung, Eliminieren “überflüssiger” Konstruktorköpfe).

Es darf allerdings bezweifelt werden, daß eine weniger naive Implementierung von OINC bzw. NCA große Geschwindigkeitsgewinne bringt, da durch den “frühen” Nichtdeterminismus viele Choice-Points angelegt werden müssen.
Außerdem ist das Anlegen neuer Gleichungen sehr aufwendig, wie bereits geschildert wurde.
Im Fall von NCA ergibt sich ein nicht unerheblicher Overhead für die Behandlung applikativer Terme. Wie diese Terme geschickter codiert werden können, ist zunächst nicht offensichtlich.

Kapitel 6

Zusammenfassung und Ausblick

Diese Arbeit beschäftigte sich mit dem Vergleich von Narrowing-Verfahren erster (Needed Narrowing und Outside-In Narrowing Calculus) und höherer Ordnung (Needed Narrowing mit Warren's Methode und Narrowing Calculus for Applicative Systems).

Das Ziel dieses Vergleich liegt vor allem in der Bereitstellung einer Argumentationsgrundlage für oder gegen die Verwendung eines der Kalküle als Grundlage funktional-logischer Programmiersprachen.
Er ermöglicht aber auch die Vereinfachung des Nachweises von Eigenschaften eines der genannten Verfahren durch Reduktion von Aussagen über ein anderes (wie beispielsweise die Unabhängigkeit OINC-errechneter Lösungen bei induktiv-sequentiellen TES).

6.1 Narrowing erster Ordnung

Es ist gelungen, konstruktiv die Simulationsbeziehung von NN durch OINC und für erfolgreiche Ableitungen durch "Reduktion der Rückrichtung" auch von OINC durch NN aufzuzeigen: Jeder NN-Ableitung kann eindeutig eine OINC-Ableitung zugeordnet werden und bei erfolgreichen Ableitungen auch umgekehrt. Der Beweis ist insofern konstruktiv, als er nicht abstrakt im Sinne einer Existenzaussage geführt wird, sondern das Aussehen der korrespondierenden Ableitungen konkret angibt. Dem Verfasser ist keine Arbeit bekannt, die eine solche konstruktive Zuordnung vornimmt.
Das Wissen über die Form der sich entsprechenden Ableitungen ermöglicht es, Voraussagen über das Verhalten der Kalküle in einer Implementierung zu treffen. Das Verhalten einer Implementierung von OINC unterscheidet sich von einer NN-Implementierung insofern, als die anzuwendenden Regeln früher (in bezug auf die gesamte Ableitung) geraten werden müssen (don't-know-Nichtdeterminismus). Daraus folgt zum einen das zeitaufwendige Anlegen einer

höheren Anzahl von Choice-Points und zum anderen die Existenz einer höheren Anzahl schließlich fehlschlagender Berechnungen. Daraus resultiert ein zentrales Resultat dieser Arbeit: Für induktiv-sequentielle TES ist NN der vorzuziehende Kalkül!
Der formale Unterschied beider Kalküle (NN als "Funktion", OINC als Inferenzschema) ist Grundlage eines weiteren Sachverhalts, der das Verhalten von OINC gegenüber NN ungünstiger abschneiden läßt: Das Anlegen neuer Gleichungen, das teure Listenoperationen erfordert.

Experimentelle Resultate bestätigen diese Aussagen.
Natürlich sind solche Benchmarks immer abhängig von der gewählten Implementierung. Die Implementierung von NN beispielsweise basiert auf (nach PROLOG) compiliertem Code; OINC wird im Rahmen dieser Arbeit als Interpreter realisiert. Die getroffenen Feststellungen könnten somit durch die Angabe einer Realisierung von OINC, die die angesprochenen Punkte geschickter als die hier verwendete behandelt, widerlegt werden.
Das ist allerdings unwahrscheinlich. OINC arbeitet als outside-in-Verfahren die Terme von oben nach unten ab. Um das zu frühe Raten von Regeln zu verhindern, wäre ein Lookahead auf die tiefer liegenden Positionen nötig. Nicht zuletzt die Definition durch ein Inferenzschema, das ständig neue Gleichungen erzeugt, läßt diesen Lookahead aus Effizienzgründen ausscheiden.

Gelänge die Transformation des Inferenzschemas in eine Funktion (wie bei NN), so müßten die genannten Aspekte neu diskutiert werden. Eine solche Funktion würde der NN-Berechnungsfunktion λ vermutlich stark ähneln.

Die diskutierten Punkte müssen bzgl. der Ausdrucksstärke der behandelten TES relativiert werden. NN rechnet auf induktiv-sequentiellen TES, OINC auf einer echten Obermenge davon: orthogonalen TES.
Die Programmiersprache Curry (z.B. mit der zugrundeliegenden operationalen Semantik des Parallel Narrowing) gestattet das Schreiben konstruktorbasierter, fast-orthogonaler Programme, wiederum einer echten Obermenge der orthogonalen TES. Schränkt man Parallel Narrowing auf orthogonale konstruktorbasierte Systeme ein, so wäre der Vergleich mit OINC auf derselben Systemklasse ein interessantes Forschungsthema.
Die Praxis zeigt, daß nicht-konstruktorbasierte, orthogonale Programme nicht eben häufig auftreten. Relativierend muß natürlich davon ausgegangen werden, daß die Möglichkeit des Schreibens solcher Programme auch das tatsächliche Schreiben solcher Programme begünstigt.

Die angestrebte Verwendung von OINC im Kontext der funktional-logischen Programmiersprachen resultiert in einer Erweiterung des Kalküls: s-OINC. s-OINC codiert die Regeln für die strikte Gleichheit in den Kalkül hinein. Die weiter oben angesprochenen Nachteile von OINC lassen sich im Fall des Narrowing von Termen, an deren Wurzelposition nicht die strikte Gleichheit steht, auf s-OINC übertragen. Der Einsatz von s-OINC sollte allerdings nicht ohne weiteres dem Einsatz von OINC vorgezogen werden. Wenn eine geringe Anzahl strikter Gleichungen erzeugt wird, wirkt sich der zusätzliche Overhead durch die neuen Inferenzregeln kontraproduktiv aus. Außerdem ist davon auszugehen, daß in realistischen Programmen die strikte Gleichheit auch auf rechten Regelseiten auftritt (z.B. für bestimmte $if - then - else$-Konstrukte). Damit muß die strikte Gleichheit dann doch als Funktion implementiert werden, und die (noch nicht einmal immer vorhandenen) Vorteile von s-OINC gegenüber OINC werden durch den Overhead durch die neuen Inferenzregeln in das Gegenteil verkehrt.

Zusammenfassend läßt sich also festhalten, daß das Rechenverhalten von NN günstiger als das von OINC und s-OINC ist, wenn man sich auf induktiv-sequentielle Systeme beschränkt.

Der Unterschied der zugrundeliegenden TES-Klassen ist ein weiterer Ausgangspunkt für eine Fortsetzung des Studiums der betrachteten Kalküle.
Als Fortsetzung dieser Arbeit wären Untersuchungen über

- das Verhalten von OINC gegenüber Parallel Narrowing auf konstruktorbasierten OTES, die auch Optimalitätseigenschaften wie Unabhängigkeit der resultierenden Substitutionen einschließen könnte,
- die Häufigkeit des Auftretens nicht-konstruktorbasierter, orthogonaler Programme in der Praxis,
- eine eventuelle Erweiterung von OINC auf fast orthogonale Systeme (bei Konstruktorbasiertheit) mit Korrektheits- und Vollständigkeitsresultaten,
- die Möglichkeit, OINC nicht durch ein Inferenzschema, sondern eine Berechnungsfunktion zu definieren, woraus sich Performancegewinne durch eine spätere Regelanwendung ergeben könnten
- den Vergleich fehlschlagender Ableitungen (d.h. eine “Bisimulation”) und
- eine Vereinfachung des sehr technischen Simulationsbeweises evtl. durch eine Transformation von NN in ein Inferernzschema

von Interesse.

6.2 Narrowing höherer Ordnung

Da sich das Konzept der Funktionen als first-class-citizens in der Welt der funktionalen Programmiersprachen durchgesetzt hat und häufige Anwendung findet, bietet sich die Suche nach einer Fortsetzung dieses Konzepts im funktional-logischen Kontext an. Das bedeutet, daß Variablen an partiell applizierte Funktionen gebunden werden können.

Intuitiv ist dazu die Verwendung der Unifikation höherer Ordnung (für eine entscheidbare Klasse von TES) nötig. Funktional-logische Sprachen wie HO-Babel, SFL oder TOY und die logische Sprache λ-Prolog verfolgen diesen Ansatz.
Tatsächlich aber kann man (natürlich unter Verlust der Ausdrucksstärke) auf Unifikation höherer Ordnung verzichten und stattdessen mit geeigneten syntaktischen Konstrukten – applikative TES – oder syntaktischen Transformationen – Warren's Methode – im Rahmen der Unifikation erster Ordnung bleiben.

In dieser Arbeit wurden NCA und NN mit Warren's Methode gegenübergestellt. Die Simulation von NN durch NCA wurde nachgewiesen – ein Resultat, daß als Nachweis der Korrektheit von Warren's Methode aufgefaßt werden kann. Ein wechselseitiges Simulationsresultat wie im Fall erster Ordnung liegt nahe, wurde aber nicht formal nachgewiesen, wenn auf rechten Regelseiten partiell applizierte Funktionen vorkommen dürfen. Verbietet man solche Regeln, ergab sich eine gegenseitige Simulierbarkeit
Der Grund dafür liegt im wesentlichen darin, daß die betrachteten TES unterschiedlich sind. Die mit der Warren'schen Methode transformierten TES enthalten für jedes Symbol der Signatur eine Anzahl neuer Regeln. Daraus folgt insbesondere, daß eine einzelne NCA-Inferenz i.a. durch eine Menge von NN-Schritten simuliert wird. Andere Ansätze wurden mit dem Ergebnis diskutiert, daß die genannte Schwierigkeit dort ebenfalls zu Problemen führen würde.

Auch wenn eine *wechselseitige* Simulationsbeziehung nicht formal nachgewiesen wurde, wurde dennoch argumentiert, warum diese bestehen müßte.
Darauf aufbauend wurden Aufwandsabschätzungen vorgenommen, die die Vermutung zulassen, daß das rechnerische Verhalten von NCA gegenüber dem von NN mit Warren's Methode weniger effizient sein dürfte. Grund ist wie im Fall erster Ordnung das zu frühe nichtdeterministische Anwenden von Regeln.
Experimentelle Resultate bestätigten diese Einschätzung.

Es wurde gezeigt, daß das Verhalten von NCA im Fall erster Ordnung dem von OINC entspricht, wenn die applikativen TES in nicht-applikativer Schreibweise aufgefaßt werden. Da in der Praxis Konstrukte höherer Ordnung vermutlich seltener als solche erster Ordnung auftreten, ergibt sich allein aus dieser Tatsache, daß NN mit Warren's Methode wegen der effizienteren Berechnungen dem Kalkül NCA vorzuziehen ist. NN in Kombination mit durch die Warren'sche Transformation eingeführten Regeln verhält sich im Fall erster Ordnung so, als ob diese neuen Regeln nicht existierten.

Auch hier stellte sich die Frage nach der Ausdrucksstärke der verwendeten TES. NCA rechnet auf applikativen orthogonalen Programmen. NN ist wiederum auf induktiv-sequentielle Systeme beschränkt (die Warren'sche Transformation verändert diese Eigenschaft nicht). Die erzielten Ergebnisse sind immer auch unter diesem Gesichtspunkt zu betrachten. Für die Diskussion sei hier auf den letzten Abschnitt verwiesen.

Narrowing höherer Ordnung kann effizienter gestaltet werden, wenn man die Funktionen mit ihren Typen annotiert. Die Implementierung eines solchen getypten Narrowing stellt eine interessante Fortsetzung des Studiums des Narrowing höherer Ordnung dar.

Die Problematik der im Fall höherer Ordnung i.a. auftretenden unendlichen Berechnungen wurde diskutiert. Insbesondere wurde gezeigt, wie die Existenz von Regeln für die strikte Gleichheit bei der gewählten Implementierung von NCA (mit PROLOG's Tiefensuche) eine unendliche Berechnung durchführt, ohne vorher ein Resultat geliefert zu haben.
Außerdem existieren i.a. nicht nur unendliche Berechnungen, sondern auch unendliche Lösungsmengen. Um Narrowing höherer Ordnung verwenden zu können, sind deshalb ausgefeilte Suchoperationen beispielsweise mit der eingekapselten Suche vonnöten.

Anlaß zu weiteren Überlegungen liefern somit die folgenden Punkte:

- der formale Nachweis der Bisimulationsbeziehung – oder der Nachweis, das sie nicht besteht – beider Kalküle, in der auch nicht erfolgreiche Ableitungen berücksichtigt würden,
- wie im Fall von OINC die Transformations des Inferenzschemas NCA durch eine Funktion oder umgekehrt – damit könnte sich neben einer Effizienzsteigerung auch der Bisimulationsnachweis vereinfachen,
- die Untersuchung ausdrucksstärkerer Klassen von TES (vgl. OINC),
- die Implementierung beider Kalküle unter Verwendung von Typinformationen und
- eine Adaptation von Suchmechanismen an diesen speziellen Fall i.a. unendlicher Lösungsmengen.

Inwieweit die Bindung von Variablen an partielle Funktionen tatsächlich ein notwendiges Feature funktional-logischer Sprachen ist, muß die Praxis zeigen. Es wurde ein nicht-triviales Beispiel (die Berechnung von Homomoprhismen zwischen Gruppen) angegeben. In jedem Fall ist davon auszugehen, daß die Bereitstellung solcher Mechanismen die Programmierer auch einlädt, diese zu verwenden.

Abkürzungsverzeichnis

Abkürzung	steht für
ATES	Applikatives TES
DT	Definierender Baum
KBO-TES	konstruktorbasiertes orthogonales TES
LOI	leftmost-outside-in
LR-DT	Links-rechts definierender Baum
NCA	narrowing calculus for applicative term rewriting systems
NN	Needed Narrowing
OI	outside-in
OINC	outside-in narrowing calculus
OTES	orthogonales TES
pDT	partieller definierender Baum
SNC-	standard narrowing calculus-
TES	Termersetzungssystem
WAM	Warren's Abstract Machine

Symbolverzeichnis

Symbol	Bemerkung
$\cdot$	Konkatenation von Positionen
$\circ$	Komposition von Substitutionen
$\square$	leere Gleichungsmenge
$\_$	anonyme Variable
$\top$	generisch ein oder mehrere *true*s; Gleichung $true \approx true$
$\triangleq$	"Entsprechung"
$\approx$	OINC-Gleichheit: "teste Unifikationsgleichheit modulo TES"
$=$	syntaktische (reflexive) Gleichheit; Gleichheit von Substitutionen als $\sigma = \vartheta \iff \sigma \leq \vartheta \wedge \vartheta \leq \sigma$
$\equiv$	strikte Gleichheit
$<, \leq$	Standardordnung auf $\mathbb{N}$, Subsumtionsordnung auf Substitutionen
$\prec, \preceq$	partielle Ordnung auf Positionen (Präfixe)
$\overset{\text{OI}}{\triangleright}$	Vereinfachungsordnung auf Narrowingableitungen
$\to_{\mathcal{R}}$	Termersetzungsrelation bzgl. $\mathcal{R}$
$\leadsto$	Narrowing-Schritt
$\looparrowright$	Multischritt
$\mapsto$	Variablenbindung in Substitutionen
$\to$	Regel $l \to r$; Reduktionsrelation
$\to_\beta$	β-Reduktion des λ-Kalküls
$t\vert_p$	Unterterm von t an der Stelle p
$t[s]_p$	t mit Ersetzung von $t\vert_p$ durch s
$\sigma[\mathcal{X}]$	Substitution σ eingeschränkt auf $\mathcal{X}$
$\mathcal{C}$	Konstruktoren; $\mathcal{C} \subseteq \Sigma$
$\mathcal{F}$	Funktionssymbole; $\mathcal{F} \subseteq \Sigma$
$\mathcal{O}(t)$	Positionen in t
$\overline{\mathcal{O}}(t)$	nichtvariable Positionen in t
$\mathcal{R}, \mathcal{S}$	Regeln eines TES oder ein TES
$\mathcal{R}_+$	$\mathcal{R} \cup \{x \approx x \to true\}$
$\mathcal{R}^{@}$	TES $\mathcal{R}$ erweitert um die Regeln, die mit der Warren'schen Methode erzeugt werden.

$\mathcal{R}^{@}_{\equiv}$	TES $\mathcal{R}^{@}$ erweitert um Regeln für $\equiv$ und diejenigen Regeln, die aus der Anwendung der Warren'schen Methode auf die Regeln für $\equiv$ entstehen.
$\mathcal{T}$	Definierender Baum
$\mathcal{T}(\Sigma, \mathcal{X})$	Terme über Σ und den Variablen aus $\mathcal{X}$
$\mathcal{V}ar(t)$	Variablen in t
$\mathcal{X}, \mathcal{W}$	Variablen
λ, λ'	NN-Berechnungsfunktion
τ, τ^{-1}	Transformationsfunktion (Warren's Methode)
$\zeta, \eta, \vartheta, \sigma, \psi, \ldots$	Substitutionen
ϱ	Variablenumbenennung
π	Muster eines DT-Knotens oder Instanz eines solchen
Δ, Σ, Γ	λ'-Aufruffolgen
Σ	Signatur, $\Sigma = \mathcal{F} \cup \mathcal{C}$
$Arity(f)$	Stelligkeit von $f \in \Sigma$
$Left(p)$	Positionen links von p (Bezug auf Term aus Kontext)
$Left(t, p)$	Positionen links von p in t
$leftmgu^{p}(t, s)$	Unifikator von s und t für die Positionen links von p
$mgu(t, s)$	allgemeinster Unifikator von s und t
$Pos_{\top}(t, s, p, q)$	Position des Funktionssymbols, das am nächsten über p in t und über q in s liegt und in beiden Termen vorkommt
$R_i^{\lambda}, R_i^{\lambda'}$	Regeln ($i \leq 4$) der Definition für λ und λ
$Right(p)$	Positionen rechts von p (Bezug auf Term aus Kontext)
$Right(t, p)$	Positionen rechts von p in t
$rightmgu^{p}(t, s)$	Unifikator von s und t für die Positionen rechts von p
$rule(v)$	Menge der $rule$-Knoten unter einem Knoten v
$Subst$	Menge der Substitutionen
l, L	linke Regelseite
r, R	rechte Regelseite
o, p, q, v	Positionen
s, t, u	Terme
x, y, z	Variablen
$\overrightarrow{x_k}$	$x_1, \ldots, x_k$
$\overline{x_k}$	$x_1 \ldots x_k$

Tabellenverzeichnis

Tabelle	Bemerkung	Seite
1	Implementierungen von NN und OINC im Vergleich: Zeiten	73
2	Profil der Berechnungen aus Tabelle 1	73
3	Profil der Berechnungen aus Tabelle 1 mit kleineren Zahlenwerten	74
4	Anzahl Choice-Points von NN und OINC	74
5	optimierter NN-Code: Anzahl Choice-Points; kein term-sharing	75
6	Implementierungen von NN und s-OINC im Vergleich: Choice-Points und Zeiten	83
7	Implementierungen von NN+Warren und NCA im Vergleich: Choice-Points und Zeiten	125
8	Profil der Berechnunges aus Tabelle 7	125
9	NN und NCA im Vergleich: Berechnung von Homomorphismen zwischen Gruppen	127
10	Profil der Berechnungen aus Tabelle 9	127

Literaturverzeichnis

[AEH94a] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Proc. 21st ACM Symposium on Principles of Programming Languages (POPL'94)*, pages 268–279, 1994.

[AEH94b] S. Antoy, R. Echahed, and M. Hanus. An optimal narrowing strategy. Draft, 1994.

[AEH97] S. Antoy, R. Echahed, and M. Hanus. Parallel Evaluation Strategies for Functional Logic Languages. In *Proc. 14th Intl. Conf. on Logic Programming (ICLP'97)*, pages 138–152. MIT Press, July 1997.

[Aït90] H. Aït-Kaci. An Overview of LIFE. In *Proc. Workshop on Next Generation Information System Technology*, Springer LNCS 504, pages 42–58, 1990. Zitiert nach [Han97].

[ALN87] H. Aït-Kaci, P. Lincoln, and R. Nasr. Le Fun: Logic, Equations, and Functions. In *Proc. 4th IEEE Intl. Symposium on Logic Programming*, pages 17–23, 1987. Zitiert nach [Han97].

[Ant92] S. Antoy. Definitional trees. In *Proc. 3rd Intl. Conf. on Algebraic and Logic Programming (ALP'92)*, Springer LNCS 632, pages 143–157, 1992.

[AT98] S. Antoy and A. Tolmach. Typed Higher-order Narrowing without Higher-order Strategies. Technical Report TR 98-2, Portland State University, 1998.

[BA95] F. Baader and C.A. Albayrak. *Termersetzungssysteme*. Aachener Beiträge zur Informatik Band 12. Verlag der Augustinus Buchhandlung, Aachen, 1995.

[BBA96] Z.O. Bayram, B.R. Bryant, and Ü. Altýnay. ROSE: A Practical Higher-Order Functional/Logic Language. In *Proc. 11th Intl. Symposium on Computer and Information Sciences*, pages 713–721, 1996.

[BBH97] J.B. Bell, F. Bellegarde, and J. Hook. Type-driven defunctionalization. In *Proc. ACM SIGPLAN Intl. Conf. on Functional Programming (ICFP'97)*, pages 25–37, 1997.

[BE86] D. Bert and R. Echahed. Design and Implementation of a Generic, Logic and Functional Programming Language. In *Proc. European Symposium on Programming*, Springer LNCS 213, pages 119–132, 1986. Zitiert nach [Han97].

[BG86] P.G. Bosco and E. Giovanetti. IDEAL: An ideal deductive applicative language. In *Proc. IEEE Intl. Symp. on Logic Programming*, pages 89–94, 1986.

[CER90] M.H.M. Cheng, M.H. van Emden, and B.E. Richards. On Warren's method for functional programming in logic. In D.H.D. Warren and P. Szeredi, editors, *Proc. 7th Intl. Conf. on Logic Programming (ICLP'90)*, pages 546–560, 1990.

[Fri85] L. Fribourg. SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting. In *Proc. IEEE Intl. Symposium on Logic Programming*, pages 172–184, 1985. Zitiert nach [Han97].

[GHLR96] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas, and M. Rodríguez-Artalejo. A Rewriting Logic for Declarative Programming. In *Proc. ESOP'96*, 1996.

[GHR92] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo. On the Completeness of Narrowing as the Operational Semantics of Functional Logic Programming. In *Proc. CSL'92*, Springer LNCS 702, pages 216–230, 1992.

[GHR97] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo. A Higher Order Rewriting Logic for Functional Logic Programming. In *Proc. 14th. Intl. Conf. on Logic Programming (ICLP'97)*, 1997.

[GLMP91] E. Giovanetti, G. Levi, C. Moiso, and C. Palamidessi. K-LEAF: A Logic plus Functional language. *J. of Computer and System Sciences*, 42:139–185, 1991.

[Gon93] J.C. González-Moreno. A correctness proof for Warren's HO into FO translation. In *Proc. GULP'93*, pages 569–585, Gizzeria Lido, Italy, 1993.

[Han90] M. Hanus. Compiling Logic Programs with Equality. In *Proc. 2nd Intl. Workshop on Programming Language Implementation and Logic Programming*, Springer LNCS 631, pages 387–401, 1990.

[Han92] M. Hanus. Implementierung logischer Programmiersprachen, 1992. Vorlesungsskript, Max-Planck-Institut für Informatik, Saarbrücken.

[Han94] M. Hanus. The integration of functions into logic programming: From theory to practice. *J. Logic Programming*, 19,20:583–628, 1994.

[Han95] M. Hanus. Efficient translation of lazy functional logic programs into PROLOG. In *Proc. 5th Intl. Workshop on Logic Program Synthesis and Transformations (LOPSTR'95)*, Springer LNCS 1048, pages 252–266, 1995.

[Han97] M. Hanus. A Unified Computation Model for Functional and Logic Programming. In *Proc. 24th Annual SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, January 1997.

[Han98] M. Hanus(ed.). Curry: An integrated functional logic language. Draft, February 1998.

[HKM95] M. Hanus, H. Kuchen, and J.J. Moreno-Navarro. Curry: A Truly Functional Logic Language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, December 1995.

[HL91] G. Huet and J.-J. Levy. Computations in orthogonal rewriting systems. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in honor of Alan Robinson*, pages 395–414. The MIT-Press, 1991.

[HL96] M. Hanus and S. Lucas. A denotational semantics for needed narrowing. In *Proc. Joint Conf. on Decl. Prog. (APPIA-GULP-PRODE'96)*, pages 259–270, 1996.

[HP96] M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. In *Proc. 7th Intl. Conf. on Rewriting Techniques and Applications (RTA'96)*, Springer LNCS 1103, pages 138–152, 1996.

[IN94] T. Ida and K. Nakahara. Leftmost outside-in narrowing calculi. Technical Report ISE-TR-94-107, University of Tsukuba, 1994.

[IN97] T. Ida and K. Nakahara. Leftmost outside-in narrowing calculi. *J. Functional Programming*, 7(2):129–161, March 1997.

[KA96] H. Kuchen and J. Anastasiadis. Higher Order Babel: Language and Implementation. In *Proc. Workshop Extensions of Logic Programming*, Springer LNAI 1050, pages 193–207, 1996.

[Kuc95] H. Kuchen. A Functional Logic Language Based on Higher Order Narrowing. In *Proc. 1995 Glasgow Functional Programming Workshop*, Workshops in Computing. Springer Verlag, 1995. ISBN 3-540-14580-X.

[Llo94] J.W. Lloyd. Combining Functional and Logic Programming Languages. In *Proc. Intl. Logic Programming Symposium*, pages 43–57, 1994. Zitiert nach [Han97].

[Llo95] J.W. Lloyd. Declarative Programming in Escher. Technical Report CSTR-95-013, University of Bristol, 1995. Zitiert nach [Han97].

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[MR92] J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: The Language BABEL. *Journal of Logic Programming*, 12:191–223, 1992.

[Nag90] M. Nagl. *Softwaretechnik: Methodisches Programmieren im Großen.* Springer Compass. Springer Verlag, 1990. ISBN 3-540-52705-2.

[Nai91] L. Naish. Adding equations to NU-Prolog. In *Proc. 3rd. Intl. Symposium on Programming Language Implementation and Logic Programming (PLILP'91)*, Springer LNCS 528, pages 15–26, 1991. Zitiert nach [Han97].

[ND94] G. Nadathur and D.Miller. Higher-Order Logic Programming. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 500–590. Oxford University Press, 1994.

[NM88] G. Nadathur and D. Miller. An Overview of λ-Prolog. In *Proc. 5th Intl. Conf. on Logic Programming (ICLP'88)*, pages 810–827. MIT Press, 1988.

[NMI95] K. Nakahara, A. Middeldorp, and T. Ida. A complete narrowing calculus for higher-order functional logic programming. In *Proc. 7th Intl. Symposium on Programming Languages, Implementations, Logics and Programs (PLILP'95)*, pages 97–114, 1995.

[Nol95] T. Noll. *Klassen applikativer Programmschemata und ihre Berechnungsstärke.* Shaker Verlag, Aachen, 1995. Dissertation.

[Pre95a] C. Prehofer. A Call-by-Need Strategy for Higher-Order Functional-Logic Programming. In *Proc. ILPS'95*, pages 147–161. MIT Press, 1995.

[Pre95b] C. Prehofer. Higher-Order Narrowing with Convergent Systems. In *Proc. 4th Intl. Conf. on Algebraic Methodology and Software Technology (AMAST'95)*, Springer LNCS 936, July 1995.

[Qia94] Z. Qian. Higher-Order Equational Logic Programming. In *Proc. 21st ACM Symposium on Principles of Programming Languages (POPL'94)*, pages 254–267, 1994.

[Rey72] J.C. Reynolds. Definitional interpreters for higher-order programming languages. In *ACM National Conference*, pages 717–740. ACM, 1972. Zitiert nach [BBH97].

[SIC96] Swedish Institute of Computer Science, The Intelligent Systems Laboratory. *SICStus Prolog User's Manual*, October 1996. Release 3#5.

[Smo95] G. Smolka. The Oz Programming Model. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, Springer LNCS 1000, pages 324–343, 1995. Zitiert nach [Han97].

[War82] D.H.D. Warren. Higher-order extensions to PROLOG: are they needed? *Machine Intelligence*, 10:441–454, 1982.