Termersetzungssysteme

Jürgen Giesl

Sommersemester 2011

Lehr- und Forschungsgebiet Informatik 2

RWTH Aachen

Inhaltsverzeichnis

Kapitel 1

Einführung

Termersetzungssysteme dienen zum Rechnen und automatischen Beweisen mit Gleichungen. Sie sind daher grundlegend für viele Bereiche der Informatik und haben unter anderem folgende Einsatzgebiete:

- Spezifikation von Programmen
  - Termersetzungssysteme werden zur Formulierung von algebraischen Spezifikationen verwendet. Der Vorteil hierbei ist, dass die Spezifikationen direkt ausführbar sind und sich hervorragend zur maschinellen Verarbeitung eignen.
  - Insbesondere setzt man Termersetzungssysteme zur Beschreibung von abstrakten Datentypen ein. Hierbei stellt sich unter anderem die Frage, welche Aussagen aus den Axiomen des abstrakten Datentyps folgen. Termersetzungssysteme bieten hocheffiziente Verfahren zum rechnergestützten Nachweis und zur Lösung solcher Gleichungen.
- Automatisierte Analyse und Verifikation von Programmen
  - Termersetzungssysteme stellen Techniken zur Verfügung, um grundlegende Fragestellungen über Programme zu untersuchen (wie z.B. Terminierung, Eindeutigkeit, Korrektheit von Programmen).
- Ausführung von Programmen
  - Termersetzungssysteme stellen eine vollständige Programmiersprache dar. Dies ist die Basis-Programmiersprache, die allen funktionalen Programmiersprachen zugrunde liegt.
  - Sie werden daher in der Implementierung vieler Programmiersprachen eingesetzt.
- Symbolisches Rechnen
  - Termersetzungsbasierte Verfahren zur Bearbeitung, Lösung und zum Nachweis von Gleichungen sind auch die Grundlage von Systemen zum symbolischen Rechnen und zur Computer-Algebra.

Als Beispiel betrachten wir eine Datenstruktur zur Darstellung der natürlichen Zahlen, wie sie in funktionalen Programmiersprachen üblich ist. Hierzu verwendet man eine nullstellige Funktion $\mathcal{O}$ und eine einstellige Funktion succ. Diese beiden Funktionen werden als *Konstruktoren* der Datenstruktur der natürlichen Zahlen benutzt, d.h., man kann mit ihnen alle Datenobjekte der Datenstruktur darstellen. Hierbei steht $\mathcal{O}$ für die Zahl 0 und succ für die Nachfolgerfunktion ("successor"). Die Zahl 1 wird also als $\mathsf{succ}(\mathcal{O})$ dargestellt, die Zahl 2 wird als $\mathsf{succ}(\mathsf{succ}(\mathcal{O}))$ dargestellt, etc.

Auf dieser Datenstruktur kann man nun z.B. den folgenden Additionsalgorithmus definieren.

$$\mathsf{plus}(\mathcal{O}, y) \equiv y \tag{1.1}$$
$$\mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathsf{plus}(x, y)) \tag{1.2}$$

Dies ist ein funktionales Programm, das durch zwei Gleichungen spezifiziert wird. Es bedeutet, dass plus das zweite Argument y zurückliefert, falls das erste Argument die Zahl $\mathcal{O}$ ist. Ansonsten ruft sich plus rekursiv auf, wobei das erste Argument um eins erniedrigt wird und das zweite Argument unverändert bleibt. Das Endresultat wird dabei um eins erhöht.

"Berechnungen" mit solch einem Programm verlaufen durch "Anwenden" der Gleichungen von links nach rechts. (Hierbei darf man beliebige Teilausdrücke anhand der Gleichungen ersetzen.) Wenn man also "$2 + 1$" berechnen möchte, ruft man plus mit den Argumenten $\mathsf{succ}(\mathsf{succ}(\mathcal{O}))$ und $\mathsf{succ}(\mathcal{O})$ auf. Die Auswertung verläuft dann wie folgt:

$$\begin{aligned}\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), \mathsf{succ}(\mathcal{O})) &\to \mathsf{succ}(\mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(\mathcal{O}))) \\ &\to \mathsf{succ}(\mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O})))) \\ &\to \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})))\end{aligned}$$

Man erhält also das Ergebnis 3, wie erwünscht.

Systeme von Gleichungen wie oben, die von links nach rechts angewendet werden sollen, bezeichnet man als *Termersetzungssysteme*. Hierbei schreibt man üblicherweise anstelle des Gleichheitszeichens "$\equiv$" einen Pfeil "$\to$", um zu verdeutlichen, in welcher Richtung die Gleichungen anzuwenden sind. Ausdrücke wie $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), \mathsf{succ}(\mathcal{O}))$ oder $\mathsf{plus}(\mathsf{succ}(x), y)$ heißen *Terme*. Terme sind aus *Variablen* (wie x und y) und *Funktionssymbolen* (wie $\mathcal{O}$, succ und plus) aufgebaut. Die Auswertung von Termen anhand von Gleichungen wie oben bezeichnet man als *Termersetzung*. Man erkennt also, dass Termersetzung ein sehr einfacher und natürlicher Mechanismus ist, der vielen Programmiersprachen zugrunde liegt, und dass die Sprache der Termersetzungssysteme in der Tat eine (bzw. *die*) grundlegende funktionale Programmiersprache ist.

Um die Zuverlässigkeit und Korrektheit von Programmen zu garantieren, ist es nicht hinreichend, sie nur zu testen, sondern es werden Techniken benötigt, um sie formal zu analysieren und zu verifizieren. Solche Analysen und Verifikationen sind jedoch im allgemeinen sehr aufwendig — insbesondere bei umfangreichen Programmen, wie sie in der Praxis eingesetzt werden. Aus diesem Grund wird versucht, diese Aufgaben weitgehend automatisch durchzuführen. Hierzu erweisen sich Termersetzungssysteme als äußerst wichtiges Werkzeug.

Folgende Fragestellungen sind typisch bei der Programmanalyse und -verifikation und in der Vorlesung werden wir Techniken kennen lernen, um diese Fragen automatisch zu untersuchen:

- *Ist das Resultat eines Programms immer eindeutig (**Konfluenz**)?*
  Die Frage ist hierbei, ob man trotz verschiedener möglicher Anwendungen von Gleichungen stets zu demselben Ergebnis kommt. Im obigen Programm ist dies der Fall. So kann man zwar zur Auswertung des Terms

  $$\mathsf{plus}(\mathcal{O}, \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathcal{O}))$$

  Gleichungen auf verschiedene Weise anwenden. Es stellt sich aber heraus, dass das Programm trotzdem deterministisch ist, d.h., das Ergebnis zum Schluss ist immer das Gleiche.

  Im allgemeinen ist dies aber natürlich nicht garantiert. Hierzu ergänzen wir das Programm um die folgende Gleichung.

  $$\mathsf{plus}(\mathsf{succ}(x), y) \;\equiv\; \mathsf{plus}(x, \mathsf{succ}(y))$$

  Auf *Grund*termen (d.h., Termen ohne Variablen, die nur aus plus, succ, und $\mathcal{O}$ bestehen) bleibt die Konfluenz erhalten, aber auf beliebigen Termen nicht. So kann man nun $\mathsf{plus}(\mathsf{succ}(x), y)$ sowohl zu $\mathsf{plus}(x, \mathsf{succ}(y))$ als auch zu $\mathsf{succ}(\mathsf{plus}(x, y))$ auswerten und danach ist keine weitere Auswertung möglich. Die Auswertung hat also zwei unterschiedliche Ergebnisse. Wir werden Verfahren kennen lernen, die die Konfluenz eines Programms automatisch untersuchen.

- *Hält ein Programm immer nach endlich vielen Schritten an (**Terminierung**)?*
  Dies bedeutet, dass das Programm keine unendlichen Auswertungsfolgen haben darf. Im obigen Beispiel erkennt man, dass bei jedem rekursiven Aufruf das erste Argument kleiner wird. Bei der Auswertungsfolge für "2+1" ist das erste Argument zunächst 2 (d.h. $\mathsf{succ}(\mathsf{succ}(\mathcal{O}))$), dann 1 ($\mathsf{succ}(\mathcal{O})$) und zuletzt 0 (d.h. $\mathcal{O}$). Da Argumente nie unendlich oft kleiner werden können, bedeutet das, dass immer nur endlich viele rekursive Aufrufe möglich sind. Daher terminiert der Algorithmus plus.

  Würde man hingegen das Programm um die weitere Gleichung

  $$\mathsf{plus}(x, y) \;\equiv\; \mathsf{plus}(y, x)$$

  ergänzen, so würde das Programm nicht mehr terminieren, denn es gibt (unter anderem) die folgende unendliche Auswertung:

  $$\mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathcal{O}) \to \mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O})) \to \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathcal{O}) \to \mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O})) \to \ldots$$

  Wir werden Techniken vorstellen, die die Terminierung von Programmen automatisch nachweisen können.

- *Erfüllt ein Programm seine Spezifikation (**Korrektheit**)?*
  Die Frage hierbei ist, ob gewünschte Aussagen über ein Programm wahr sind (oder

ob aus einer Menge von Axiomen bestimmte andere Aussagen folgen). Beispielsweise kann man untersuchen, ob die Aussagen

$$\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x)) \tag{1.3}$$
$$\mathsf{plus}(\mathsf{plus}(x, y), z) \equiv \mathsf{plus}(x, \mathsf{plus}(y, z)) \tag{1.4}$$

für das ursprüngliche Additionsprogramm mit den Gleichungen (1.1) und (1.2) (bei beliebigen x, y, z) gilt. Die zweite Aussage bedeutet, dass wir überprüfen wollen, ob dieses Programm eine assoziative Funktion berechnet.

Ähnliche Fragen treten auch bei der Spezifikation von Programmen auf. Eine algebraische Spezifikation von Datentypen besteht aus der Angabe von bestimmten Axiomen und die Frage ist, welche Aussagen über den Datentyp aus den Axiomen folgen. Als Beispiel betrachten wir die Axiome, die *Gruppen* spezifizieren:

$$\mathsf{f}(x, \mathsf{f}(y, z)) \equiv \mathsf{f}(\mathsf{f}(x, y), z) \tag{1.5}$$
$$\mathsf{f}(x, \mathsf{e}) \equiv x \tag{1.6}$$
$$\mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e} \tag{1.7}$$

Hierbei ist f eine assoziative Verknüpfung, e ist das rechtsneutrale Element und i ist die rechtsinverse Funktion. Ein Beispiel für eine Gruppe sind z.B. die ganzen Zahlen, wobei f die Addition, e die Zahl 0 und i die Funktion ist, die zu einer Zahl n jeweils die Zahl $-n$ berechnet. Hier stellt sich z.B. die Frage, ob in jeder Gruppe die inverse Funktion "selbstinvers" ist, d.h., ob eine zweimalige Anwendung von i wieder das ursprüngliche Element erzeugt. Mit anderen Worten, man will wissen, ob die Gleichung

$$\mathsf{i}(\mathsf{i}(n)) \equiv n \tag{1.8}$$

für alle Elemente n aus den obigen drei Axiomen folgt. Diese Frage bezeichnet man als *Wortproblem*. Dies ist tatsächlich der Fall, wie man durch eine (ziemlich lange und komplizierte) Folge von Anwendungen dieser Gleichungen zeigen kann (siehe Bsp. 3.1.15, S. 25). Wir werden zeigen, wie man mit Hilfe von Termersetzungssystemen solch eine Frage automatisch lösen kann.

- *Wie kann man ein unvollständiges Programm automatisch* ***vervollständigen****?*
  Hierbei wird die Frage untersucht, wie man aus einem nur unvollständig angegebenen Programm (bzw. aus einer Menge von Axiomen) ein vollständiges (konfluentes und terminierendes) Programm automatisch generieren kann. Beispielsweise ist das Programm aus den beiden plus-Gleichungen (1.1) und (1.2) zusammen mit

  $$\mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{plus}(x, \mathsf{succ}(y))$$

  ja nicht mehr konfluent. Man kann es aber (automatisch) um die weitere Gleichung

  $$\mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$$

  ergänzen und erhält dadurch ein "äquivalentes" Programm, das sowohl konfluent als auch terminierend ist. Die Frage ist, wie man solche Vervollständigungen automatisch findet.

Die Vorlesung gliedert sich in die folgenden Kapitel:

1. Einführung
Hier wird eine kurze Übersicht über das behandelte Gebiet gegeben und die Bedeutung der untersuchten Fragestellungen motiviert.

2. Grundlagen
In diesem Kapitel führen wir die Syntax und Semantik von Termen und Gleichungen ein.

3. Termersetzung und Deduktion von Gleichungen
Nun stellen wir Methoden vor, um Aussagen über Gleichungen automatisch nachzuweisen. Insbesondere wird das zentrale Konzept der Termersetzungssysteme eingeführt, das auch den weiteren Kapiteln zugrundeliegt. Hiermit erhält man ein sehr leistungsfähiges Verfahren, um Gleichheiten zu berechnen bzw. automatisch zu beweisen. Die Voraussetzung hierfür ist allerdings, dass die Termersetzungssysteme konfluent (d.h., deterministisch) sind und terminieren. Wenn man zu einem gegebenen Gleichungssystem ein solches äquivalentes Termersetzungssystem konstruieren kann, so stellt dieses Termersetzungssystem ein Verfahren zur Verfügung, das (automatisch) entscheidet, welche Gleichungen aus dem Gleichungssystem folgen. Auf diese Weise lassen sich z.B. Aussagen wie (1.3) oder (1.8) automatisch nachweisen.

4. Terminierung von Termersetzungssystemen
Wie im vorigen Kapitel erläutert, ist man an terminierenden und konfluenten Termersetzungssystemen interessiert. Hier wird die Frage untersucht, wie man (automatisch) herausfindet, ob ein Termersetzungssystem terminiert, d.h., ob es nur endliche Berechnungen zulässt.

5. Konfluenz von Termersetzungssystemen
Nun werden Verfahren vorgestellt, die automatisch untersuchen, ob ein Termersetzungssystem konfluent ist, d.h., ob das Ergebnis der Berechnung immer eindeutig ist.

6. Vervollständigung von Termersetzungssystemen
Dieses Kapitel behandelt ein Verfahren, um zu einer gegebenen Menge von Gleichungen ein konfluentes und terminierendes Termersetzungssystem zu konstruieren. Auf diese Weise lassen sich Entscheidungsverfahren für Gleichungssysteme gewinnen.

Für manche Aussagen über Programme (wie z.B. (1.4)) ist ein Induktionsbeweis erforderlich. Wir zeigen, dass man hierfür ebenfalls das Vervollständigungsverfahren verwenden kann. Dadurch entsteht eine Methode für automatische Induktionsbeweise, die solche Aussagen verifizieren kann.

Für weitere Literatur zum Thema "Termersetzungssysteme" sei auf [Ave95, BA95, BN98, Bün98, DJ90, Ohl02, Ter03, Wal95] verwiesen, die diesem Skript auch teilweise zugrunde lagen. Ich danke René Thiemann, Benedikt Bollig, Carsten Fuhs und Marc Brockschmidt für ihre konstruktiven Kommentare und Vorschläge beim Korrekturlesen des Skripts.

Kapitel 2

Grundlagen

In diesem Kapitel legen wir die Sprache der Terme und Gleichungen fest, die wir im Folgenden zur Formulierung von Aussagen und von Programmen (Termersetzungssystemen) verwenden werden. Hierzu verwenden wir eine Teilmenge der Prädikatenlogik 1. Stufe. Gleichungen entsprechen dann sogenannten universellen atomaren Formeln der Prädikatenlogik. In Abschnitt 2.1 wird die Syntax von Termen und Gleichungen erläutert und in Abschnitt 2.2 stellen wir die Semantik vor, d.h., wir definieren, was es bedeutet, dass eine Gleichung aus einer Menge von Axiomen *folgt*. Diese beiden Abschnitte dienen vor allem auch dazu, um die im Folgenden verwendeten Notationen einzuführen.

2.1 Syntax von Gleichungssystemen

Die *Syntax* legt fest, aus welchen Zeichenreihen die Ausdrücke einer Sprache bestehen. Wir betrachten die Sprache der Gleichungen zwischen Termen. Hierzu definieren wir zunächst das Alphabet, aus dem Terme (d.h., Ausdrücke unserer Sprache) gebildet werden.

Definition 2.1.1 (Signatur) *Eine* Signatur $\Sigma = \bigcup_{i \in \mathbb{N}} \Sigma_i$ *ist eine Vereinigung von paarweise disjunkten endlichen Mengen* Σ_i*. Jedes* $f \in \Sigma_i$ *heißt* i*-stelliges* Funktionssymbol*. Die Elemente von* Σ_0 *werden auch* Konstanten *genannt. Wir verlangen stets, dass* Σ *endlich ist und dass* $\Sigma_0 \neq \varnothing$ *gilt.*

Beispiel 2.1.2 Ein Beispiel für eine Signatur ist $\Sigma_0 = \{\mathcal{O}\}$, $\Sigma_1 = \{\mathsf{succ}\}$, $\Sigma_2 = \{\mathsf{plus}, \mathsf{times}\}$.

Nun definieren wir, wie man Datenobjekte in unserer Sprache repräsentiert. Der Einfachheit halber betrachten wir nur den unsortierten Fall (d.h., Terme sind nicht *typisiert*). Die Konzepte der Termersetzung lassen sich aber leicht auf Sorten erweitern.

Definition 2.1.3 (Term) *Sei* Σ *eine Signatur und* $\mathcal{V}$ *eine nicht-leere (abzählbar) unendliche Menge mit* $\mathcal{V} \cap \Sigma = \varnothing$*. Die Elemente von* $\mathcal{V}$ *heißen* Variablen*.*

$\mathcal{T}(\Sigma, \mathcal{V})$ *bezeichnet die Menge aller* Terme *(über* Σ *und* $\mathcal{V}$*). Hierbei ist* $\mathcal{T}(\Sigma, \mathcal{V})$ *die kleinste Teilmenge von* $(\Sigma \cup \mathcal{V} \cup \{\,(\,,\,)\,,\text{“,”}\,\})^*$ *mit*

(1) $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$ *und*

(2) $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$*, falls* $f \in \Sigma_n$*,* $n \geq 0$ *und* $t_i \in \mathcal{T}(\Sigma, \mathcal{V})$ *für alle* $i \in \{1, \ldots, n\}$*.*

$\mathcal{T}(\Sigma)$ *steht für* $\mathcal{T}(\Sigma, \varnothing)$*, d.h., für die Menge aller variablenfreien Terme (oder* Grundterme*).*

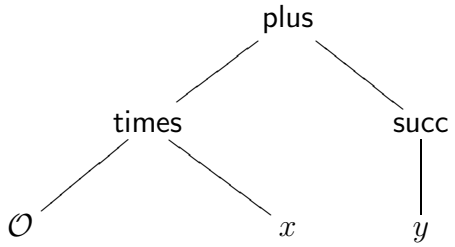
Für einen Term t *bezeichnet* $\mathcal{V}(t)$ *die Menge aller Variablen in* t *und für* $T \subseteq \mathcal{T}(\Sigma, \mathcal{V})$ *definieren wir* $\mathcal{V}(T) = \bigcup_{t \in T} \mathcal{V}(t)$*.*

Ein Term q *ist ein* Teilterm *von* t *(Schreibweise* $t \trianglerighteq q$*) gdw.* $t = q$ *oder* $t = f(t_1, \ldots, t_n)$ *und* $t_i \trianglerighteq q$ *für ein* $1 \leq i \leq n$*. Die Terme* $t_1, \ldots, t_n$ *sind die* direkten *Teilterme von* t*. Ein Teilterm* q *von* t *ist ein* echter Teilterm *(Schreibweise* $t \triangleright q$*) gdw.* $t \neq q$*.*

Beispiel 2.1.4 Wir betrachten die Signatur Σ aus Bsp. 2.1.2. Wenn $\mathcal{V} = \{x, y, z, \ldots\}$ ist, dann erhalten wir z.B.:

$$\begin{aligned} \mathcal{O}, \mathsf{succ}(\mathcal{O}), \mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O})) &\in \mathcal{T}(\Sigma), \\ \mathsf{plus}(\mathsf{times}(\mathcal{O}, x), \mathsf{succ}(y)) &\in \mathcal{T}(\Sigma, \mathcal{V}). \end{aligned}$$

Terme lassen sich auch als (Vielweg-)Bäume darstellen. Der Term $\mathsf{plus}(\mathsf{times}(\mathcal{O}, x), \mathsf{succ}(y))$ entspricht dann dem folgenden Baum.



Nun legen wir fest, wie Gleichungen in unserer Sprache gebildet werden.

Definition 2.1.5 (Gleichungen) *Für* $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{V})$ *heißt ein Ausdruck*

$$t_1 \equiv t_2$$

(Term-)gleichung *(über* Σ *und* $\mathcal{V}$*). Das Symbol "*$\equiv$*" wird also als* syntaktisches *Gleichheitszeichen benutzt. (Die Aussage "*$t_1 \equiv t_2$*" muss also nicht "wahr" sein!) Eine Menge* $\mathcal{E}$ *von Gleichungen wird* Termgleichungssystem *(über* Σ *und* $\mathcal{V}$*) genannt. Eine Gleichung ist also nichts anderes als ein Paar von Termen (wobei man meist* $s \equiv t$ *anstelle von* (s, t) *schreibt) und ein Gleichungssystem ist eine beliebige Teilmenge von* $\mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$*.*

Die Sprache der Termgleichungen ist eine formale Sprache, in der mathematische Sachverhalte, aber auch Fakten des täglichen Lebens formuliert werden können. Daher werden mit Gleichungssystemen (mathematische) Theorien *axiomatisiert*. Im Folgenden werden wir immer nur endliche Gleichungssysteme $\mathcal{E}$ betrachten; die meisten Resultate lassen sich aber auch auf unendliche Gleichungssysteme übertragen.

Beispiel 2.1.6 Sei Σ wieder die Signatur aus Bsp. 2.1.2. Die Gleichungen

$$\begin{aligned} \mathsf{plus}(\mathcal{O}, y) &\equiv y \\ \mathsf{plus}(\mathsf{succ}(x), y) &\equiv \mathsf{succ}(\mathsf{plus}(x, y)) \end{aligned}$$

axiomatisieren die Additionsfunktion auf natürlichen Zahlen, wenn succ als die Nachfolgerfunktion aufgefasst wird. Man erkennt bereits den Zusammenhang zur funktionalen Programmierung, denn diese beiden Gleichungen sind im Prinzip ein funktionales Programm. In der Sprache HASKELL könnte man dieses Programm bereits direkt ausführen. Welche Aussagen aus diesen Axiomen folgen und inwieweit solche Programme ausführbar sind, wird im Folgenden noch deutlich werden.

Beispiel 2.1.7 Sei $\Sigma_0 = \{\mathsf{e}\}$, $\Sigma_1 = \{\mathsf{i}\}$, $\Sigma_2 = \{\mathsf{f}\}$ und $\mathcal{V} = \{x, y, z, \ldots\}$. Wir betrachten die folgenden Termgleichungen:

$$\mathsf{f}(x, \mathsf{f}(y, z)) \equiv \mathsf{f}(\mathsf{f}(x, y), z) \tag{2.1}$$
$$\mathsf{f}(x, \mathsf{e}) \equiv x \tag{2.2}$$
$$\mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e} \tag{2.3}$$
$$\mathsf{f}(x, y) \equiv \mathsf{f}(y, x) \tag{2.4}$$

Dann axiomatisiert $\{(2.1)\}$ alle Halbgruppen (d.h., alle Mengen mit einer assoziativen binären Operation), $\{(2.1), (2.2), (2.3)\}$ axiomatisiert alle Gruppen und $\{(2.1)-(2.4)\}$ axiomatisiert alle abelschen Gruppen.

2.2 Semantik von Gleichungssystemen

Eine Axiomatisierung "definiert" in gewisser Weise ein Programm oder eine mathematische Struktur (z.B. eine Gruppe). Man interessiert sich nun für Aussagen über dieses Programm bzw. Aussagen, die in dieser Struktur gelten. Aussagen werden ebenfalls durch Termgleichungen dargestellt. Beispielsweise kann man fragen, ob die Aussage

$$\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$$

für das Additionsprogramm aus Bsp. 2.1.6 gilt. Ebenso kann man in Bsp. 2.1.7 fragen, ob die Aussage

$$\mathsf{i}(\mathsf{i}(n)) \equiv n$$

in jeder Gruppe gilt (d.h., ob die inverse Funktion immer selbstinvers ist).

Vor Beantwortung solcher Fragen müssen wir festlegen, was es bedeutet, dass eine Aussage aus bestimmten Axiomen folgt. Bislang haben wir nur erläutert, wie Gleichungen gebildet werden. Wir haben jedoch noch nicht definiert, was solche Gleichungen eigentlich *aussagen*. Dazu muss man den Gleichungen eine *Semantik*, d.h. eine Bedeutung, zuordnen.

Wir werden zunächst die Semantik von Termen definieren. Hierzu verwendet man sogenannte *Interpretationen*, die eine Menge von Objekten $\mathcal{A}$ festlegen und jedem (syntaktischen) Funktionssymbol f eine Funktion α_f und jeder Variablen $x \in \mathcal{V}$ ein Objekt aus $\mathcal{A}$ zuordnen.

Definition 2.2.1 (Interpretation, Algebra) *Für eine Signatur Σ ist eine* Σ-Interpretation *ein Tripel $I = (\mathcal{A}, \alpha, \beta)$. Die Menge $\mathcal{A}$ ist der* Träger *der Interpretation, wobei stets $\mathcal{A} \neq \varnothing$ gilt.*

Weiter ist $\alpha = (\alpha_f)_{f \in \Sigma}$ mit $\alpha_f : \underbrace{\mathcal{A} \times \ldots \times \mathcal{A}}_{n-\text{mal}} \to \mathcal{A}$ für $f \in \Sigma_n$. Die Funktion α_f heißt die Deutung *des Funktionssymbols f unter der Interpretation I.*

Die Abbildung $\beta : \mathcal{V} \to \mathcal{A}$ heißt Variablenbelegung *für die Interpretation I. Eine Variablenbelegung ordnet jeder Variablen $x \in \mathcal{V}$ ein Element $\beta(x) \in \mathcal{A}$ zu.*

Zu jeder Interpretation I erhält man eine Funktion $I : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathcal{A}$ wie folgt:

$$\begin{aligned} I(x) &= \beta(x) \text{ für alle } x \in \mathcal{V} \\ I(f(t_1, \ldots, t_n)) &= \alpha_f(I(t_1), \ldots, I(t_n)) \text{ für alle } f \in \Sigma \text{ und } t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V}) \end{aligned}$$

Man nennt $I(t)$ die Interpretation des Terms t *unter der Interpretation I.*

Eine Σ-Interpretation ohne Variablenbelegung $(\mathcal{A}, \alpha)$ wird als Σ-Algebra *bezeichnet. Eine Algebra besitzt also eine Trägermenge und den Funktionssymbolen aus Σ werden Funktionen auf dieser Trägermenge zugeordnet.*

Beispiel 2.2.2 Wir betrachten wieder die Signatur Σ aus Bsp. 2.1.2. Eine Interpretation für diese Signatur ist beispielsweise $I = (\mathcal{A}, \alpha, \beta)$ mit

$$\begin{aligned} \mathcal{A} &= \mathbb{N} = \{0, 1, 2, \ldots\}, \\ \alpha_{\mathcal{O}} &= 0, \\ \alpha_{\mathsf{succ}}(n) &= n + 1, \\ \alpha_{\mathsf{plus}}(n, m) &= n + m, \\ \alpha_{\mathsf{times}}(n, m) &= n \cdot m, \\ \beta(x) &= 5, \\ \beta(y) &= 3, \\ &\vdots \end{aligned}$$

Dann ist $I(\mathsf{plus}(\mathsf{succ}(x), y)) = \alpha_{\mathsf{plus}}(\alpha_{\mathsf{succ}}(\beta(x)), \beta(y)) = 9$.

Eine weitere Interpretation ist $I' = (\mathcal{A}, \alpha, \beta')$ mit $\beta'(x) = 2$ und $\beta'(y) = 1$. Daher folgt $I'(\mathsf{plus}(\mathsf{succ}(x), y)) = 4$.

Schließlich betrachten wir auch noch eine dritte Interpretation $I'' = (\mathcal{A}'', \alpha'', \beta)$ mit $\mathcal{A}'' = \mathbb{Q}$, $\alpha''_{\mathcal{O}} = 0$, $\alpha''_{\mathsf{succ}}(x) = x + 1$, $\alpha''_{\mathsf{times}}(x, y) = x \cdot y$ und $\alpha''_{\mathsf{plus}}(x, y) = \frac{x}{y}$, falls $y \neq 0$ und $\alpha''_{\mathsf{plus}}(x, 0) = 0$. (Die Deutungen von Funktionssymbolen müssen immer *totale* Funktionen sein.) Es gilt $I''(\mathsf{plus}(\mathsf{succ}(x), y)) = \frac{6}{3} = 2$.

Nun können wir die Semantik von Gleichungen definieren, d.h., wir legen fest, welche Gleichungen $t_1 \equiv t_2$ wahr sind unter einer Interpretation I bzw. unter einer Algebra A. Man sagt dann, I bzw. A *erfüllt* die Gleichung $t_1 \equiv t_2$ (oder "I bzw. A ist ein *Modell* von $t_1 \equiv t_2$").

Definition 2.2.3 (Erfüllen einer Gleichung, Modell) *Eine Interpretation* $I = (\mathcal{A}, \alpha, \beta)$ erfüllt *eine Gleichung* $t_1 \equiv t_2$*, geschrieben "*$I \models t_1 \equiv t_2$*", gdw.* $I(t_1) = I(t_2)$.

Eine Algebra $A = (\mathcal{A}, \alpha)$ *erfüllt* $t_1 \equiv t_2$ *(*$A \models t_1 \equiv t_2$*) gdw.* $I \models t_1 \equiv t_2$ *für alle Interpretationen der Form* $I = (\mathcal{A}, \alpha, \beta)$*. Für alle Variablenbelegungen* β *müssen also die Deutungen von* t_1 *und* t_2 *identisch sein. Man sagt dann auch, eine Algebra ist ein* Modell *einer Gleichung. Analog ist eine Algebra* A *ein Modell einer Gleichungsmenge* $\mathcal{E}$ *(*$A \models \mathcal{E}$*) gdw.* $A \models t_1 \equiv t_2$ *für alle Gleichungen* $t_1 \equiv t_2$ *in* $\mathcal{E}$*.*

Aus einer Gleichungsmenge $\mathcal{E}$ folgt *die Gleichung* $t_1 \equiv t_2$ *(abgekürzt "*$\mathcal{E} \models t_1 \equiv t_2$*") gdw. für alle Algebren* A *mit* $A \models \mathcal{E}$ *gilt* $A \models t_1 \equiv t_2$*. (Das Zeichen "*$\models$*" steht also sowohl für Erfüllbarkeit durch eine Algebra bzw. Interpretation als auch für Folgerbarkeit aus einer Gleichungsmenge. Was jeweils gemeint ist, erkennt man daran, ob links des Zeichens "*$\models$*" eine Algebra bzw. Interpretation oder eine Gleichungsmenge steht.) Anstelle von "*$\varnothing \models t_1 \equiv t_2$*" schreibt man meist "*$\models t_1 \equiv t_2$*". In diesem Fall heißt die Gleichung* $t_1 \equiv t_2$ allgemeingültig *(d.h., alle Algebren sind Modell dieser Gleichung).*

Wir definieren die Relation $\equiv_{\mathcal{E}}$ *auf* $\mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$ *als* $s \equiv_{\mathcal{E}} t$ *gdw.* $\mathcal{E} \models s \equiv t$*. Die Relation* $\equiv_{\mathcal{E}}$ *beschreibt also alle Gleichheiten, die aus der Menge* $\mathcal{E}$ *folgen.*

Beispiel 2.2.4 Wir betrachten wieder die Interpretationen aus Bsp. 2.2.2. Sei $A = (\mathcal{A}, \alpha)$ (d.h., die Algebra, die der Interpretation I entspricht). Dann gilt:

$$
\begin{array}{rcl}
I & \models & \mathsf{plus}(x, y) \equiv \underbrace{\mathsf{succ}(\mathsf{succ}(\ldots(\mathsf{succ}}_{\text{8-mal "succ"}}(\mathcal{O}))\ldots)) \\
A & \not\models & \mathsf{plus}(x, y) \equiv \underbrace{\mathsf{succ}(\mathsf{succ}(\ldots(\mathsf{succ}}_{\text{8-mal "succ"}}(\mathcal{O}))\ldots)) \\
A & \models & \mathsf{plus}(x, y) \equiv \mathsf{plus}(y, x) \\
I'' & \not\models & \mathsf{plus}(x, y) \equiv \mathsf{plus}(y, x) \\
\{\mathsf{plus}(x, y) \equiv \mathsf{plus}(y, x)\} & \models & \mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O})) \equiv \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathcal{O})
\end{array}
$$

Wenn $\mathcal{E} = \{\mathsf{plus}(x, y) \equiv \mathsf{plus}(y, x)\}$, dann gilt also $\mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O})) \equiv_{\mathcal{E}} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathcal{O})$.

Beispiel 2.2.5 Sei Σ definiert wie in Bsp. 2.1.7. Sei $A = (\mathbb{Z}, \alpha)$ eine Σ-Algebra mit

$$
\begin{array}{rcl}
\alpha_{\mathsf{e}} & = & 0 \\
\alpha_{\mathsf{i}}(n) & = & -n \\
\alpha_{\mathsf{f}}(n, m) & = & n + m
\end{array}
$$

Dann gilt $A \models \{(2.1) - (2.4)\}$ und $A \models \mathsf{i}(\mathsf{i}(n)) = n$.

Sei $B = (\mathcal{B}, \alpha')$ eine Σ-Algebra mit

$$
\begin{array}{rcl}
\mathcal{B} & = & \text{Menge der linearen Listen über beliebigem Alphabet} \\
 & & \text{mit mindestens 2 Zeichen} \\
\alpha'_{\mathsf{e}} & = & [\,] \text{ (leere Liste)} \\
\alpha'_{\mathsf{i}}([a_1, \ldots, a_n]) & = & [a_n, \ldots, a_1] \text{ (Invertieren von Listen)} \\
\alpha'_{\mathsf{f}}([a_1, \ldots a_n], [b_1, \ldots, b_m]) & = & [a_1, \ldots, a_n, b_1, \ldots, b_m] \text{ (Listenkonkatenation)}
\end{array}
$$

Dann gilt $B \models \{(2.1), (2.2)\}$ und $B \models \mathsf{i}(\mathsf{i}(n)) = n$, aber $B \not\models (2.3)$ und $B \not\models (2.4)$.

Die Frage, ob $s \equiv_{\mathcal{E}} t$ für zwei Terme s und t bei einem Gleichungssystem $\mathcal{E}$ gilt, bezeichnet man als *Wortproblem*. Das Wortproblem untersucht also, ob eine Aussage $s \equiv t$ nicht nur in einer bestimmten Algebra gilt, sondern in allen Algebren, die durch gewisse gemeinsame Eigenschaften ausgezeichnet sind. Beispielsweise bedeutet

$$\mathsf{i}(\mathsf{i}(n)) \equiv_{\{(2.1)-(2.3)\}} n,$$

dass in *allen* Gruppen die inverse Funktion selbstinvers ist. Im nächsten Kapitel lernen wir Methoden kennen, um solche Fragestellungen automatisch zu untersuchen.

Kapitel 3

Termersetzung und Deduktion von Gleichungen

In diesem Kapitel untersuchen wir drei Verfahren, um das Wortproblem $s \equiv_{\mathcal{E}} t$ systematisch und ggf. automatisch zu lösen. Diese Verfahren untersuchen also, ob eine Gleichung tatsächlich aus einer Menge von Axiomen folgt. Abschnitt 3.1 stellt eine erste Methode vor, wie dieses Problem automatisch behandelt werden kann. Dieses erste (naive) Verfahren ist zwar für den praktischen Einsatz noch nicht geeignet, es stellt aber die Grundlage für die verbesserten Verfahren der nächsten Abschnitte dar. In Abschnitt 3.2 wird ein leistungsfähiges und in der Praxis verwendetes Verfahren (der sogenannte Kongruenzabschluss) präsentiert, das das Wortproblem für solche Gleichungssysteme $\mathcal{E}$ lösen kann, die keine Variablen enthalten. Dies ist ein durchaus relevantes Problem; allerdings muss man in vielen Anwendungen auch Gleichungen zwischen Termen mit Variablen betrachten. Aus diesem Grund wird in Abschnitt 3.3 das Konzept der Termersetzungssysteme betrachtet, mit dem man das Wortproblem für beliebige Gleichungssysteme untersuchen kann.

3.1 Deduktion von Gleichungen

Generell wird eine mathematische Struktur oder Theorie durch Angabe eines Gleichungssystems $\mathcal{E}$ axiomatisiert, d.h. "definiert", und man ist an Aussagen $s \equiv t$, die in solch einer Struktur gelten, interessiert. Mit anderen Worten, man möchte untersuchen, ob $\mathcal{E} \models s \equiv t$ gilt. Hierzu müssen (mit unserem bisherigen Kenntnisstand) *alle* Algebren A und *alle* Variablenbelegungen (d.h., alle Interpretationen) betrachtet werden. Die Definition der semantischen Folgerungsbeziehung "$\models$" liefert also keinen Anhaltspunkt, um systematisch (bzw. automatisch) zu überprüfen, ob $s \equiv_{\mathcal{E}} t$ gilt.

Als Abhilfe wird daher der semantischen Folgerungsbeziehung nun ein *Kalkül*, d.h. ein *Ableitungssystem*, gegenübergestellt, mit dem das Wortproblem $s \equiv_{\mathcal{E}} t$ systematisch (und automatisch) überprüft werden kann. Zur Definition des Kalküls sind weitere technische Begriffe erforderlich: Mit *Substitutionen* und *Teiltermersetzungen* werden Operationen auf Termen definiert, mit denen dann ein Ableitungsbegriff formuliert werden kann.

Wir führen zunächst den Begriff der Substitution ein. Substitutionen erlauben die Ersetzung (oder "Instantiierung") von Variablen durch Terme.

Definition 3.1.1 (Substitution und Matching) *Eine Abbildung $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$ heißt* Substitution *gdw. $\sigma(x) \neq x$ nur für endlich viele $x \in \mathcal{V}$ gilt. $SUB(\Sigma, \mathcal{V})$ bezeichnet die Menge aller Substitutionen über Σ und $\mathcal{V}$ und $id \in SUB(\Sigma, \mathcal{V})$ ist die identische Substitution (d.h., $id(x) = x$ für alle Variablen $x \in \mathcal{V}$).*

$DOM(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ heißt der Domain *von σ. Da $DOM(\sigma)$ endlich ist, ist die Substitution σ als die endliche Menge von Paaren $\{x/\sigma(x) \mid x \in DOM(\sigma)\}$ darstellbar. Die Substitution σ ist eine* Grundsubstitution *gdw. $\sigma(x) \in \mathcal{T}(\Sigma)$ für alle $x \in DOM(\sigma)$.*

Substitutionen werden homomorph zu Abbildungen $\sigma : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathcal{T}(\Sigma, \mathcal{V})$ erweitert, d.h. $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$.

Für zwei Terme s, t sagen wir, dass der Term s den Term t matcht*, falls es eine Substitution σ gibt mit $\sigma(s) = t$. Die Substitution σ heißt dann ein* Matcher *von s und t.*

Für $x^* = x_1 \ldots x_n \in \mathcal{V}^*$ mit $x_i \neq x_j$ für $i \neq j$ und $t^* = t_1 \ldots t_n \in \mathcal{T}(\Sigma, \mathcal{V})^*$ schreiben wir oft $\{x^*/t^*\}$ als Kurzschreibweise für die Substitution $\{x_1/t_1, \ldots, x_n/t_n\}$. Weiterhin schreibt man Substitutionen auch oft *hinter* den Term, auf den sie angewendet werden, d.h., $t\sigma$ steht für $\sigma(t)$.

Beispiel 3.1.2 Wir betrachten wieder die Signatur von Bsp. 2.1.2. Ein Beispiel für eine Substitution ist $\sigma = \{x/\mathsf{plus}(x, y), y/\mathcal{O}, z/\mathsf{succ}(z)\}$. Man erhält dann

$$\sigma(\mathsf{plus}(x, y)) \;=\; \mathsf{plus}(x, y)\sigma \;=\; \mathsf{plus}(\mathsf{plus}(x, y), \mathcal{O}).$$

Der Term $\mathsf{plus}(x, y)$ matcht daher den Term $\mathsf{plus}(\mathsf{plus}(x, y), \mathcal{O})$ und die Substitution σ ist ein Matcher.

Ist eine Gleichung $s \equiv t$ in einer Algebra A gültig, so gilt dies auch für jede Instanz $s\sigma \equiv t\sigma$ der Gleichung. Die Gleichung gilt also unabhängig davon, welche Elemente des Trägers den Variablen zugewiesen werden. Die Variablen in s und t sind daher implizit *allquantifiziert*. Damit bleibt die Gültigkeit einer Gleichung erhalten, wenn Variablen durch beliebige Terme ersetzt werden. Ebenso verhält es sich mit der Relation $\equiv_{\mathcal{E}}$, d.h., $\equiv_{\mathcal{E}}$ ist *abgeschlossen* oder *stabil unter Substitutionen*.

Definition 3.1.3 (Stabilität) *Eine Relation $\to$ über Termen (d.h., $\to \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$ heißt* stabil *oder* abgeschlossen unter Substitutionen*, wenn für alle Terme t_1, t_2 und alle Substitutionen σ aus $t_1 \to t_2$ folgt, dass auch $t_1\sigma \to t_2\sigma$ gilt.*

Das folgende Lemma beschreibt die oben genannten Zusammenhänge zwischen dem syntaktischen Begriff "Substitution" und dem semantischen Begriff "Variablenbelegung". Zum Beweis verwenden wir eine *strukturelle Induktion* über Terme. Dies ist möglich aufgrund der induktiven Definition von Termen (Def. 2.1.3). Wenn man eine Aussage $\varphi(t)$ für alle Terme $t \in \mathcal{T}(\Sigma, \mathcal{V})$ zeigen will, so kann man wie folgt vorgehen: Im *Induktionsanfang* zeigt man, dass die Aussage für alle Variablen gilt (d.h. $\varphi(x)$ muss für alle $x \in \mathcal{V}$ bewiesen werden) und dass die Aussage auch für alle Konstanten gilt (d.h. $\varphi(c)$ muss für alle $c \in \Sigma_0$ gelten). Im *Induktionsschluss* muss man beweisen, dass die Aussage für alle Terme der Gestalt $f(t_1, \ldots, t_n)$ (für alle Funktionssymbole $f \in \Sigma_n$ mit $n > 0$ und alle $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$)

gilt. Als *Induktionshypothese* darf man nun jeweils voraussetzen, dass die Aussage für die direkten Teilterme $t_1, \ldots, t_n$ schon gilt. Mit anderen Worten, man muss

$$\varphi(t_1) \wedge \ldots \wedge \varphi(t_n) \;\rightarrow\; \varphi(f(t_1, \ldots, t_n))$$

zeigen. Hat man sowohl den Induktionsanfang als auch den Induktionsschluss bewiesen, so folgt in der Tat $\varphi(t)$ für alle Terme t.[1] Intuitiv ist dieses Beweisprinzip korrekt, weil es direkt der Definition der Terme (Def. 2.1.3) folgt, d.h., weil $\mathcal{T}(\Sigma, \mathcal{V})$ die kleinste Menge ist mit

(1) $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$ und

(2) $f(t^*) \in \mathcal{T}(\Sigma, \mathcal{V})$, falls $f \in \Sigma$ und $t^* \in \mathcal{T}(\Sigma, \mathcal{V})^*$.

Analog lassen sich auch strukturelle Induktionsbeweise über jede andere Datenstruktur durchführen, die auf induktive Weise definiert ist. Es wird sich in Kapitel 4 herausstellen, dass dieses Beweisprinzip ein Spezialfall eines noch allgemeineren Beweisprinzips (der sogenannten *noetherschen Induktion*) ist. Dort werden wir auch die Korrektheit dieses Beweisprinzips formal nachweisen.

Lemma 3.1.4 ($\equiv_\mathcal{E}$ stabil) *Sei $I = (\mathcal{A}, \alpha, \beta)$ eine Σ-Interpretation für eine Signatur Σ, seien $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$, sei σ eine Substitution und sei $I' = (\mathcal{A}, \alpha, \beta')$ mit $\beta'(x) = I(x\sigma)$ für alle $x \in \mathcal{V}$. Dann gilt:*

(a) $I(t\sigma) = I'(t)$ (Substitutionslemma)

(b) $I \models s\sigma \equiv t\sigma$ *gdw.* $I' \models s \equiv t$

(c) Für alle Σ-Algebren A gilt: Falls $A \models s \equiv t$, dann $A \models s\sigma \equiv t\sigma$.

(d) Für alle Gleichungssysteme $\mathcal{E}$ gilt: Falls $s \equiv_\mathcal{E} t$, dann auch $s\sigma \equiv_\mathcal{E} t\sigma$. Die Relation $\equiv_\mathcal{E}$ ist also abgeschlossen unter Substitutionen.

Beweis.

(a) Der Beweis wird durch strukturelle Induktion über den Aufbau des Terms t geführt. Die Aussage $\varphi(t)$, die wir hier also betrachten, ist

> "Für alle Substitutionen σ und alle Interpretationen I, I' wie oben gilt $I(t\sigma) = I'(t)$."

Im Induktionsanfang ist t entweder eine Variable oder eine Konstante (d.h., ein Funktionssymbol aus Σ_0).

Falls t eine Variable x ist, so gilt

$$I(x\sigma) = \beta'(x) = I'(x).$$

[1] Oftmals teilt man den Induktionsbeweis auch so auf, dass man nur $\varphi(x)$ für alle Variablen x im Induktionsanfang zeigt und den Fall der Konstanten im Induktionsschluss mitbehandelt. Dort betrachtet man dann also alle Terme der Art $f(t_1, \ldots, t_n)$, wobei n auch 0 sein darf.

Falls t eine Konstante $c \in \Sigma_0$ ist, so gilt

$$I(c\sigma) = I(c) = \alpha_c = I'(c) = I'(c\sigma).$$

Im Induktionsschluss betrachten wir Terme der Art $t = f(t_1, \ldots, t_n)$ mit $n > 0$ und setzen als Induktionshypothese voraus, dass die Aussage bereits für die direkten Teilterme $t_1, \ldots, t_n$ von t gilt. Die Induktionshypothesen sind also die Aussagen $\varphi(t_1), \ldots, \varphi(t_n)$. Nun gilt

$$I(t\sigma) = I(f(t_1\sigma, \ldots, t_n\sigma)) = \alpha_f(I(t_1\sigma), \ldots, I(t_n\sigma)).$$

Aus der Induktionshypothese folgt $I(t_i\sigma) = I'(t_i)$. Man erhält daher

$$\begin{aligned} & \alpha_f(I(t_1\sigma), \ldots, I(t_n\sigma)) \\ = \; & \alpha_f(I'(t_1), \ldots, I'(t_n)) \\ = \; & I'(f(t_1, \ldots, t_n)) \\ = \; & I'(t). \end{aligned}$$

(b) Es gilt

$$\begin{aligned} I \models s\sigma \equiv t\sigma \quad & \text{gdw.} \quad I(s\sigma) = I(t\sigma) \\ & \text{gdw.} \quad I'(s) = I'(t) \\ & \text{gdw.} \quad I' \models s \equiv t \end{aligned}$$

nach Teil (a).

(c) Sei $A = (\mathcal{A}, \alpha)$. Zu zeigen ist, dass jede Interpretation $I = (\mathcal{A}, \alpha, \beta)$ (d.h., für jede Variablenbelegung β) die Gleichung $s\sigma \equiv t\sigma$ erfüllt. Zu jeder solchen Interpretation sei $I' = (\mathcal{A}, \alpha, \beta')$ mit $\beta'(x) = I(x\sigma)$ für alle $x \in \mathcal{V}$. Aus $A \models s \equiv t$ folgt $I' \models s \equiv t$, was nach Teil (b) äquivalent ist zu $I \models s\sigma \equiv t\sigma$.

(d) Sei $s \equiv_{\mathcal{E}} t$, d.h., $\mathcal{E} \models s \equiv t$. Dann gilt $A \models s \equiv t$ für alle Modelle A von $\mathcal{E}$. Nach Teil (c) gilt dann auch $A \models s\sigma \equiv t\sigma$ für alle Modelle A von $\mathcal{E}$ und demnach $\mathcal{E} \models s\sigma \equiv t\sigma$, d.h., $s\sigma \equiv_{\mathcal{E}} t\sigma$. □

Nun definieren wir, wie man die Teilterme eines Terms adressieren und gegebenenfalls ersetzen kann. Hierzu verwenden wir das Konzept der *Stellen* von Termen..

Definition 3.1.5 (Stelle) *Für einen Term t ist $Occ(t)$ die Menge aller* Stellen *(oder "Positionen") von t, wobei $Occ(t)$ die kleinste Teilmenge von $\mathbb{N}^*$ ist mit*

- $\epsilon \in Occ(t)$ *und*
- $i\,\pi \in Occ(t)$, *falls* $t = f(t_1, \ldots, t_n)$, $1 \leq i \leq n$ *und* $\pi \in Occ(t_i)$.

Für einen Term t mit $\pi \in Occ(t)$ bezeichnet $t|_\pi$ den Teilterm von t an der Stelle π, *wobei*

- $t|_\epsilon = t$ *und*
- $f(t_1, \ldots, t_n)|_{i\,\pi} = t_i|_\pi$.

Für einen Term t mit $\pi \in Occ(t)$ und einen Term r bezeichnet $t[r]_\pi$ den Term, der aus t durch Ersetzen von $t|_\pi$ durch r entsteht, d.h.

- $t[r]_\epsilon = r$ *und*
- $f(t_1, \ldots, t_n)[r]_{i\,\pi} = f(t_1, \ldots, t_i[r]_\pi, \ldots, t_n)$.

Für zwei Stellen $\pi_1, \pi_2 \in Occ(t)$ sagen wir dass π_1 unterhalb *von π_2 ist ($\pi_1 >_{\mathbb{N}^*} \pi_2$) gdw. π_2 ein echtes Anfangsstück von π_1 ist, d.h. $\pi_1 = \pi_2\,\pi$ für ein $\pi \in \mathbb{N}^+$. Zwei Stellen π_1, π_2 sind* voneinander unabhängig *($\pi_1 \perp \pi_2$) gdw. weder $\pi_1 >_{\mathbb{N}^*} \pi_2$ noch $\pi_2 >_{\mathbb{N}^*} \pi_1$ noch $\pi_1 = \pi_2$ gilt.*

Beispiel 3.1.6 Betrachten wir den Term $t = \mathsf{plus}(\mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O}))), \mathsf{succ}(\mathcal{O}))$. Dann haben wir

$$Occ(t) = \{\epsilon, 1, 11, 111, 112, 1121, 2, 21\}$$

und

$$\begin{aligned} t|_\epsilon &= \mathsf{plus}(\mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O}))), \mathsf{succ}(\mathcal{O})) \\ t|_1 &= \mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{succ}(\mathcal{O}))) \\ t|_{112} &= \mathsf{succ}(\mathcal{O}) \\ t|_{21} &= \mathcal{O} \end{aligned}$$

Damit ergibt sich

$$t[\mathsf{times}(\mathcal{O}, \mathcal{O})]_{112} = \mathsf{plus}(\mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{times}(\mathcal{O}, \mathcal{O}))), \mathsf{succ}(\mathcal{O})).$$

Wir haben $112 >_{\mathbb{N}^*} 11 >_{\mathbb{N}^*} 1$ und $111 \perp 112$.

Ist eine Gleichung $s \equiv t$ in einer Algebra A gültig, so gilt dies auch für jede Einbettung der Terme s und t in Kontexte, d.h., auch Gleichungen der Form $q[s]_\pi \equiv q[t]_\pi$ sind dann gültig. Ebenso verhält es sich mit der Relation $\equiv_\mathcal{E}$, d.h., $\equiv_\mathcal{E}$ ist nicht nur abgeschlossen unter Substitutionen, sondern auch *abgeschlossen unter Kontexten*. Solche Relationen bezeichnet man auch als *monoton*.

Definition 3.1.7 (Monotonie) *Eine Relation $\to$ über Termen heißt* monoton*, wenn für alle Terme t_1, t_2, q und alle $\pi \in Occ(q)$ aus $t_1 \to t_2$ folgt, dass auch $q[t_1]_\pi \to q[t_2]_\pi$ gilt.*

Ähnlich zur strukturellen Induktion über Terme kann man strukturelle Induktion auch über Stellen führen, denn die Datenstruktur der Stellen $\mathbb{N}^*$ ist ebenfalls induktiv definiert. Wenn man eine Aussage $\varphi(\pi)$ für alle Stellen zeigen will, so kann man wie folgt vorgehen: Im *Induktionsanfang* zeigt man, dass die Aussage für die oberste Stelle ϵ gilt. Im *Induktionsschluss* muss man beweisen, dass die Aussage für alle Positionen der Gestalt $i\pi'$ gilt. Als *Induktionshypothese* darf man nun jeweils voraussetzen, dass die Aussage für die "kürzere" Stelle π' gilt. Dieses Beweisprinzip ist korrekt, weil es direkt der Definition der Stellen $\mathbb{N}^*$ folgt, d.h., $\mathbb{N}^*$ ist die kleinste Menge mit

(1) $\epsilon \in \mathbb{N}^*$ und

(2) $i\pi' \in \mathbb{N}^*$, falls $i \in \mathbb{N}$ und $\pi' \in \mathbb{N}^*$.

Alternativ kann man im Induktionsschluss auch die Aussage für alle Positionen der Gestalt $\pi' i$ zeigen und sie jeweils für π' als Induktionshypothese voraussetzen.

Lemma 3.1.8 ($\equiv_\mathcal{E}$ monoton) *Seien $s, t, q \in \mathcal{T}(\Sigma, \mathcal{V})$ und $\pi \in Occ(q)$. Dann gilt:*

(a) Für alle Σ-Algebren A gilt: Falls $A \models s \equiv t$, dann $A \models q[s]_\pi \equiv q[t]_\pi$.

(b) Für alle Gleichungssysteme $\mathcal{E}$ gilt: Falls $s \equiv_\mathcal{E} t$, dann auch $q[s]_\pi \equiv_\mathcal{E} q[t]_\pi$. Die Relation $\equiv_\mathcal{E}$ ist also abgeschlossen unter Kontexten.

Beweis.

(a) Der Beweis wird durch strukturelle Induktion über den Aufbau der Position π geführt. Die Aussage $\varphi(\pi)$, die wir hier also betrachten, ist

"Für alle A, s, t, q mit $\pi \in Occ(q)$ gilt: Falls $A \models s \equiv t$, dann $A \models q[s]_\pi \equiv q[t]_\pi$."

Im Induktionsanfang ist $\pi = \epsilon$. Hier ist $q[s]_\epsilon = s$, $q[t]_\epsilon = t$ und somit gilt die Aussage trivialerweise.

Im Induktionsschluss betrachten wir Stellen der Art $\pi = i\pi'$ und setzen als Induktionshypothese voraus, dass die Aussage bereits für π' gilt. Da $\pi \in Occ(q)$ gelten muss, hat q die Gestalt $f(q_1, \ldots, q_i, \ldots, q_n)$ und $\pi' \in Occ(q_i)$. Damit gilt $q[s]_\pi = f(q_1, \ldots, q_i[s]_{\pi'}, \ldots, q_n)$ und $q[t]_\pi = f(q_1, \ldots, q_i[t]_{\pi'}, \ldots, q_n)$.

Sei $A = (\mathcal{A}, \alpha)$ mit $A \models s \equiv t$. Zu zeigen ist, dass für alle Interpretationen $I = (\mathcal{A}, \alpha, \beta)$ mit beliebiger Variablenbelegung β jeweils $I \models f(q_1, \ldots, q_i[s]_{\pi'}, \ldots, q_n) \equiv f(q_1, \ldots, q_i[t]_{\pi'}, \ldots, q_n)$ gilt. Dies ist gleichbedeutend mit $\alpha_f(I(q_1), \ldots, I(q_i[s]_{\pi'}), \ldots, I(q_n)) = \alpha_f(I(q_1), \ldots, I(q_i[t]_{\pi'}), \ldots, I(q_n))$. Hierzu genügt es, zu zeigen, dass $I(q_i[s]_{\pi'}) = I(q_i[t]_{\pi'})$ ist. Aus $A \models s \equiv t$ folgt aber nach der Induktionshypothese bereits $A \models q_i[s]_{\pi'} \equiv q_i[t]_{\pi'}$ und somit auch $I(q_i[s]_{\pi'}) = I(q_i[t]_{\pi'})$.

Die Induktionshypothese $\varphi(\pi')$ lautet nämlich wie folgt:

"Für alle A, s, t, q mit $\pi' \in Occ(q)$ gilt: Falls $A \models s \equiv t$, dann $A \models q[s]_{\pi'} \equiv q[t]_{\pi'}$."

Da dies *für alle* A, s, t, q gilt (diese A, s, t, q müssen also nicht mit den oben gewählten übereinstimmen), kann man $\varphi(\pi')$ umformulieren zu

"Für alle A, s, t, q' mit $\pi' \in Occ(q')$ gilt: Falls $A \models s \equiv t$, dann $A \models q'[s]_{\pi'} \equiv q'[t]_{\pi'}$."

Wählt man als q' den Term q_i, so folgt wie erwünscht $A \models q_i[s]_{\pi'} \equiv q_i[t]_{\pi'}$.

(b) Sei $s \equiv_\mathcal{E} t$, d.h., $\mathcal{E} \models s \equiv t$. Dann gilt $A \models s \equiv t$ für alle Modelle A von $\mathcal{E}$. Nach Teil (a) gilt dann auch $A \models q[s]_\pi \equiv q[t]_\pi$ für alle Modelle A von $\mathcal{E}$ und demnach $\mathcal{E} \models q[s]_\pi \equiv q[t]_\pi$, d.h., $q[s]_\pi \equiv_\mathcal{E} q[t]_\pi$. □

Um weitere Eigenschaften von Relationen wie $\equiv_{\mathcal{E}}$ beschreiben zu können, benötigen wir die folgenden bekannten Begriffe über Relationen.

Definition 3.1.9 (Äquivalenzrelation, Kongruenzrelation) *Sei M eine Menge und $\to$ eine Relation mit $\to \subseteq M \times M$ (d.h., $\to$ ist eine Relation "*über*" oder "*auf*" M). Man sagt auch, $(M, \to)$ ist ein* (abstraktes) Reduktionssystem (ARS)*. Die Relation $\to$ heißt*

- reflexiv*, falls $t \to t$ für alle $t \in M$ gilt*
- symmetrisch*, falls aus $t_1 \to t_2$ jeweils $t_2 \to t_1$ für alle $t_1, t_2 \in M$ folgt*
- transitiv*, falls aus $t_1 \to t_2$ und $t_2 \to t_3$ jeweils $t_1 \to t_3$ für alle $t_1, t_2, t_3 \in M$ folgt*
- Äquivalenzrelation*, falls $\to$ reflexiv, symmetrisch und transitiv ist.*

Die Relation $\to^=$ (die reflexive Hülle *von $\to$) ist die kleinste reflexive Relation, die $\to$ enthält, d.h., die kleinste Relation, so dass für alle $t_1, t_2 \in M$ gilt*

- *wenn $t_1 \to t_2$, dann $t_1 \to^= t_2$ und*
- *$t_1 \to^= t_1$*

Die Relation $\to^+$ (die transitive Hülle *von $\to$) ist die kleinste transitive Relation, die $\to$ enthält, d.h., die kleinste Relation, so dass für alle $t_1, t_2, t_3 \in M$ gilt*

- *wenn $t_1 \to t_2$, dann $t_1 \to^+ t_2$ und*
- *wenn $t_1 \to t_2 \to^+ t_3$, dann $t_1 \to^+ t_3$*

Die Relation $\to^*$ (die transitiv-reflexive Hülle *von $\to$) ist die kleinste transitive und reflexive Relation, die $\to$ enthält, d.h., die kleinste Relation, so dass für alle $t_1, t_2, t_3 \in M$ gilt*

- *wenn $t_1 \to^+ t_2$, dann $t_1 \to^* t_2$ und*
- *$t_1 \to^* t_1$.*

Die Relation $\leftrightarrow$ ist die symmetrische Hülle *von $\to$, d.h., es gilt $t_1 \leftrightarrow t_2$ gdw. $t_1 \to t_2$ oder $t_2 \to t_1$. Die Relation $\leftrightarrow^*$ (die* transitiv-reflexiv-symmetrische Hülle *von $\to$) ist demnach die kleinste Äquivalenzrelation, die $\to$ enthält.*

Eine Äquivalenzrelation $\to$ über Termen $\mathcal{T}(\Sigma, \mathcal{V})$ heißt Kongruenzrelation*, wenn für alle $f \in \Sigma$ und $s_1, \ldots, s_n, t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$ mit $s_1 \to t_1, \ldots, s_n \to t_n$ auch $f(s_1, \ldots, s_n) \to f(t_1, \ldots, t_n)$ gilt.*

Generell bedeutet $s \to^+ t$ also, dass es $s_0, \ldots, s_n$ mit $n > 0$ gibt, so dass

$$s = s_0 \to s_1 \to \ldots \to s_n = t$$

gilt. Die Bedeutung von $s \to^* t$ ist analog, nur ist hier $n = 0$ möglich. Wir schreiben auch $\to^n$ für eine n-fache "Hintereinanderanwendung" der Relation $\to$, d.h., $s \to^n t$ gdw. es $s_1, \ldots, s_{n-1}$ gibt mit $s \to s_1 \to s_2 \to \ldots \to s_{n-1} \to t$.

Neben der Abgeschlossenheit unter Substitutionen und Kontexten kann man nun leicht die folgende Aussage über $\equiv_{\mathcal{E}}$ zeigen.

Lemma 3.1.10 ($\equiv_\mathcal{E}$ ist Äquivalenz- und Kongruenzrelation) *Für alle Gleichungssysteme $\mathcal{E}$ ist $\equiv_\mathcal{E}$ eine Äquivalenz- und Kongruenzrelation.*

Beweis. Für alle Terme t_1, t_2, t_3 und alle Algebren A gilt:

- $A \models t_1 \equiv t_1$
- Falls $A \models t_1 \equiv t_2$, dann $A \models t_2 \equiv t_1$.
- Falls $A \models t_1 \equiv t_2$ und $A \models t_2 \equiv t_3$, dann $A \models t_1 \equiv t_3$.

Der Grund ist, dass $A \models s \equiv t$ jeweils Gleichheit von $I(s)$ und $I(t)$ unter entsprechenden Interpretationen I bedeutet und die Gleichheit ist eine Äquivalenzrelation.

Da die obigen Aussagen dann insbesondere auch für jedes Modell A von $\mathcal{E}$ gelten, ist $\equiv_\mathcal{E}$ eine Äquivalenzrelation.

Da $\equiv_\mathcal{E}$ monoton ist (Lemma 3.1.8), folgt aus $s_1 \equiv_\mathcal{E} t_1, \ldots, s_n \equiv_\mathcal{E} t_n$:

$$f(s_1, s_2, \ldots, s_n) \equiv_\mathcal{E} f(t_1, s_2, \ldots, s_n) \equiv_\mathcal{E} \ldots \equiv_\mathcal{E} f(t_1, \ldots, t_{n-1}, s_n) \equiv_\mathcal{E} f(t_1, \ldots, t_n).$$

Aus der Transitivität von $\equiv_\mathcal{E}$ ergibt sich nun auch, dass $\equiv_\mathcal{E}$ eine Kongruenzrelation ist. □

Unser Ziel ist nun, $s \equiv_\mathcal{E} t$ ohne Rückgriff auf die Semantik untersuchen zu können. Hierzu definieren wir eine rein syntaktische Ersetzungs- und Beweisrelation. Diese ermöglicht es, Aussagen wie $s \equiv_\mathcal{E} t$ auch *automatisch* zu beweisen. Die Grundlage für diesen (und die späteren) Deduktionsmechanismen ist das Prinzip, Gleiches durch Gleiches zu ersetzen.

Definition 3.1.11 (Ersetzungsrelation, Beweisrelation, Herleitung) *Für ein Gleichungssystem $\mathcal{E}$ ist die Ersetzungsrelation $\to_\mathcal{E} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$ definiert durch*

$$s \to_\mathcal{E} t \quad \textit{gdw.} \quad s|_\pi = t_1\sigma \ \textit{ und } \ t = s[t_2\sigma]_\pi$$

für eine Stelle $\pi \in Occ(s)$, eine Gleichung $t_1 \equiv t_2 \in \mathcal{E}$ und ein $\sigma \in SUB(\Sigma, \mathcal{V})$.

Die Relation $\leftrightarrow^*_\mathcal{E} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$ heißt die Beweisrelation *von $\mathcal{E}$ und bezeichnet die transitiv-reflexiv-symmetrische Hülle von $\to_\mathcal{E}$. Wir sagen, die Gleichung $s \equiv t$ ist aus $\mathcal{E}$* herleitbar *(geschrieben "$\mathcal{E} \vdash s \equiv t$"), falls $s \leftrightarrow^*_\mathcal{E} t$.*

Eine *Herleitung* besteht immer aus (null oder mehreren) $\leftrightarrow_\mathcal{E}$-Schritten. Die Aussage $s \leftrightarrow^*_\mathcal{E} t$ bedeutet also, dass es $s_0, \ldots, s_n$ (mit $n \geq 0$) gibt, so dass $s = s_0 \leftrightarrow_\mathcal{E} s_1 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$ gilt. Hierbei besagt $s_i \leftrightarrow_\mathcal{E} s_{i+1}$, dass es einen Teilterm $s_i|_\pi$ gibt, so dass eine Seite t_1 einer Gleichung $t_1 \equiv t_2$ oder $t_2 \equiv t_1$ aus $\mathcal{E}$ diesen Teilterm matcht. Es gibt also einen Matcher σ, so dass $t_1\sigma = s_i|_\pi$ ist. Der Term s_{i+1} entsteht dann aus s_i, indem dieser Teilterm durch die entsprechend instantiierte andere Seite $t_2\sigma$ der Gleichung ersetzt wird.

Beispiel 3.1.12 Es gilt $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \leftrightarrow^*_\mathcal{E} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$. Man erhält für die Gleichung $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ folgende Herleitung aus den Axiomen von Bsp. 2.1.6. Die Stellen, an denen Teilterme ersetzt werden, sind durch Unterstreichung markiert. Gleichungen werden "von links nach rechts" angewendet und werden

deshalb je nach Bedarf als $t_1 \equiv t_2$ oder $t_2 \equiv t_1$ geschrieben. Wir kürzen dabei $\mathsf{succ}(\mathsf{succ}(x))$ durch $\mathsf{succ}^2(x)$ ab, etc.

$\underline{\mathsf{plus}(\mathsf{succ}^2(\mathcal{O}), x)}$	$\mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$	$\sigma = \{x/\mathsf{succ}(\mathcal{O}), y/x\}$
$\mathsf{succ}(\underline{\mathsf{plus}(\mathsf{succ}(\mathcal{O}), x)})$	$\mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$	$\sigma = \{x/\mathcal{O}, y/x\}$
$\mathsf{succ}^2(\underline{\mathsf{plus}(\mathcal{O}, x)})$	$\mathsf{plus}(\mathcal{O}, y) \equiv y$	$\sigma = \{y/x\}$
$\mathsf{succ}(\underline{\mathsf{succ}(x)})$	$y \equiv \mathsf{plus}(\mathcal{O}, y)$	$\sigma = \{y/\mathsf{succ}(x)\}$
$\underline{\mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{succ}(x)))}$	$\mathsf{succ}(\mathsf{plus}(x, y)) \equiv \mathsf{plus}(\mathsf{succ}(x), y)$	$\sigma = \{x/\mathcal{O}, y/\mathsf{succ}(x)\}$
$\mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$		

Man erkennt hierbei auch, dass die *Namen* der Variablen in den Gleichungen $\mathcal{E}$ sowohl für die Relation $\equiv_\mathcal{E}$ wie für die Beweisrelation $\leftrightarrow^*_\mathcal{E}$ unerheblich sind. Der Grund ist, dass die Variablen implizit als allquantifiziert betrachtet werden. Die Gleichung $\mathsf{plus}(\mathcal{O}, y) \equiv y$ könnte also auch durch $\mathsf{plus}(\mathcal{O}, z) \equiv z$ ersetzt werden.

Die Relation $\leftrightarrow^*_\mathcal{E}$ ist auf rein syntaktische Weise definiert und somit bedeutet "$\mathcal{E} \vdash s \equiv t$", dass die Gleichung $s \equiv t$ mit syntaktischen Mitteln aus $\mathcal{E}$ hergeleitet wird. Hingegen bedeutet "$\mathcal{E} \models s \equiv t$" bzw. "$s \equiv_\mathcal{E} t$", dass die Gleichung $s \equiv t$ in allen Modellen von $\mathcal{E}$ gilt. Dies ist daher eine semantische Aussage, die sich so nicht direkt überprüfen lässt. Unser Ziel ist daher, das Wortproblem mit Hilfe der Beweisrelation $\leftrightarrow^*_\mathcal{E}$ anstelle von $\equiv_\mathcal{E}$ zu überprüfen. Hierzu muss man natürlich untersuchen, inwieweit $s \leftrightarrow^*_\mathcal{E} t$ und $s \equiv_\mathcal{E} t$ zusammenhängen.

Wir wissen bereits, dass $\equiv_\mathcal{E}$ eine Kongruenzrelation ist, die sowohl unter Substitutionen wie auch unter Kontexten abgeschlossen ist. Dies trifft offensichtlich auch auf $\leftrightarrow^*_\mathcal{E}$ zu.

Lemma 3.1.13 ($\leftrightarrow^*_\mathcal{E}$ stabile und monotone Kongruenzrelation) *Sei $\mathcal{E}$ ein Gleichungssystem, seien $s, t, q \in \mathcal{T}(\Sigma, \mathcal{V})$, $\pi \in Occ(q)$ und σ eine Substitution. Dann ist $\leftrightarrow^*_\mathcal{E}$ eine Äquivalenz- und Kongruenzrelation und es gilt:*

(a) Falls $s \to_\mathcal{E} t$, dann auch $s\sigma \to_\mathcal{E} t\sigma$ und falls $s \leftrightarrow^*_\mathcal{E} t$, dann auch $s\sigma \leftrightarrow^*_\mathcal{E} t\sigma$. Die Relationen $\to_\mathcal{E}$ und $\leftrightarrow^*_\mathcal{E}$ sind also abgeschlossen unter Substitutionen.

(b) Falls $s \to_\mathcal{E} t$, dann auch $q[s]_\pi \to_\mathcal{E} q[t]_\pi$ und falls $s \leftrightarrow^*_\mathcal{E} t$, dann auch $q[s]_\pi \leftrightarrow^*_\mathcal{E} q[t]_\pi$. Die Relationen $\to_\mathcal{E}$ und $\leftrightarrow^*_\mathcal{E}$ sind also abgeschlossen unter Kontexten.

Beweis. Die Reflexivität und Transitivität folgen sofort aus der Definition, da $\leftrightarrow^*_\mathcal{E}$ die transitiv-reflexive Hülle von $\leftrightarrow_\mathcal{E}$ ist. Die Relation $\leftrightarrow_\mathcal{E}$ ist nach Definition symmetrisch. Dann ist auch $\leftrightarrow^*_\mathcal{E}$ symmetrisch, denn aus $s \leftrightarrow^*_\mathcal{E} t$ folgt, dass es $s_0, \ldots, s_n$ (mit $n \geq 0$) gibt, so dass $s = s_0 \leftrightarrow_\mathcal{E} s_1 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$. Aufgrund der Symmetrie von $\leftrightarrow_\mathcal{E}$ folgt dann $t = s_n \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_1 = s$, d.h., $t \leftrightarrow^*_\mathcal{E} s$. (Formal ist hier eine Induktion über die Länge der Herleitung nötig.) Somit ist $\leftrightarrow^*_\mathcal{E}$ eine Äquivalenzrelation.

(a) Wir zeigen zunächst, dass $\to_\mathcal{E}$ unter Substitutionen abgeschlossen ist. Aus $s \to_\mathcal{E} t$ folgt, dass es ein $\tau \in Occ(s)$ und eine Substitution θ gibt mit $s|_\tau = t_1\theta$ und $t = s[t_2\theta]_\tau$ für eine Gleichung $t_1 \equiv t_2$ aus $\mathcal{E}$. Damit gilt $s\sigma|_\tau = s|_\tau\sigma = t_1\theta\sigma$ und $t\sigma = s[t_2\theta]_\tau\sigma = s\sigma[t_2\theta\sigma]_\tau$. Wenn σ' die Substitution $\theta\sigma$ ist (d.h., die Komposition der beiden Substitutionen, bei der erst θ und dann σ angewendet wird), dann folgt also

$s\sigma|_\tau = t_1\sigma'$ und $t\sigma = s\sigma[t_2\sigma']_\tau$. Somit erhält man $s\sigma \to_\mathcal{E} t\sigma$. Damit folgt auch, dass $\leftrightarrow_\mathcal{E}$ unter Substitutionen abgeschlossen ist.

Nun beweisen wir, dass $\leftrightarrow^*_\mathcal{E}$ unter Substitutionen abgeschlossen ist. Aus $s \leftrightarrow^*_\mathcal{E} t$ folgt wieder, dass es $s_0, \ldots, s_n$ (mit $n \geq 0$) gibt, so dass $s = s_0 \leftrightarrow_\mathcal{E} s_1 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$. Durch Induktion über n (d.h., durch Induktion über die Länge der Herleitung) zeigen wir nun, dass dann auch $s\sigma = s_0\sigma \leftrightarrow_\mathcal{E} s_1\sigma \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n\sigma = t\sigma$ gilt. Der Induktionsanfang ($n = 0$) ist trivial. Im Induktionsschluss ist $n > 0$ und aus der Induktionshypothese folgt $s\sigma = s_0\sigma \leftrightarrow_\mathcal{E} s_1\sigma \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_{n-1}\sigma$. Da $\leftrightarrow_\mathcal{E}$ unter Substitutionen abgeschlossen ist, folgt aus $s_{n-1} \leftrightarrow_\mathcal{E} s_n = t$ auch $s_{n-1}\sigma \leftrightarrow_\mathcal{E} s_n\sigma = t\sigma$. Somit erhält man in der Tat $s\sigma \leftrightarrow^*_\mathcal{E} t\sigma$.

(b) Wir zeigen zuerst, dass $\to_\mathcal{E}$ unter Kontexten abgeschlossen ist. Aus $s \to_\mathcal{E} t$ folgt, dass es ein $\tau \in Occ(s)$ und eine Substitution θ gibt mit $s|_\tau = t_1\theta$ und $t = s[t_2\theta]_\tau$ für eine Gleichung $t_1 \equiv t_2$ aus $\mathcal{E}$. Damit gilt $q[s]_\pi|_{\pi\tau} = t_1\theta$ und $q[t]_\pi = q[s]_\pi[t_2\theta]_{\pi\tau}$. Wenn π' die Stelle $\pi\tau$ ist, dann folgt also $q[s]_\pi|_{\pi'} = t_1\theta$ und $q[t]_\pi = q[s]_\pi[t_2\theta]_{\pi'}$. Somit erhält man $q[s]_\pi \to_\mathcal{E} q[t]_\pi$. Damit folgt auch, dass $\leftrightarrow_\mathcal{E}$ unter Kontexten abgeschlossen ist.

Der Beweis, dass auch $\leftrightarrow^*_\mathcal{E}$ unter Kontexten abgeschlossen ist, ist nun vollkommen analog zum Beweis von (a) (durch Induktion über die Länge der Herleitung von $s \leftrightarrow^*_\mathcal{E} t$).

Dass $\leftrightarrow^*_\mathcal{E}$ eine Kongruenzrelation auf $\mathcal{T}(\Sigma, \mathcal{V})$ ist, folgt analog zum Beweis von Lemma 3.1.10 aus der Monotonie und der Transitivität. □

Nun können wir das grundlegende Resultat zeigen, dass die syntaktisch definierte Relation $\leftrightarrow^*_\mathcal{E}$ mit der semantisch definierten Relation $\equiv_\mathcal{E}$ identisch ist. Mit anderen Worten, man kann eine Gleichung $s \equiv t$ genau dann aus $\mathcal{E}$ herleiten ($\mathcal{E} \vdash s \equiv t$), wenn sie auch aus $\mathcal{E}$ folgt ($\mathcal{E} \models s \equiv t$). Man kann daher $\leftrightarrow^*_\mathcal{E}$ auch als *operationale Semantik* der $\mathcal{E}$-Gleichheit betrachten. Ein Verfahren, dass $s \leftrightarrow^*_\mathcal{E} t$ überprüft, ist also korrekt und vollständig (d.h., es beweist genau die wahren Gleichungen). Dieser wichtige Satz, der die Grundlage für viele der weiteren Entwicklungen darstellt, wurde von Birkhoff 1935 gezeigt.

Satz 3.1.14 (Satz von Birkhoff [Bir35]) *Sei $\mathcal{E}$ ein Gleichungssystem. Dann sind die Relationen $\equiv_\mathcal{E}$ und $\leftrightarrow^*_\mathcal{E}$ identisch, d.h., es gilt $\mathcal{E} \models s \equiv t$ gdw. $\mathcal{E} \vdash s \equiv t$ für alle Terme s und t.*

Beweis.

Korrektheit:

Wir zeigen zunächst, dass aus $s \leftrightarrow_\mathcal{E} t$ folgt, dass auch $s \equiv_\mathcal{E} t$ gilt. Es existiert also eine Gleichung $t_1 \equiv t_2$ oder $t_2 \equiv t_1$ in $\mathcal{E}$, so dass $s|_\pi = t_1\sigma$ und $t = s[t_2\sigma]_\pi$ für eine Substitution σ und eine Stelle $\pi \in Occ(s)$. Aus der Symmetrie von $\equiv_\mathcal{E}$ (Lemma 3.1.10) folgt $t_1 \equiv_\mathcal{E} t_2$. Da $\equiv_\mathcal{E}$ stabil ist (Lemma 3.1.4 (d)), gilt auch $t_1\sigma \equiv_\mathcal{E} t_2\sigma$ und da $\equiv_\mathcal{E}$ darüber hinaus monoton ist (Lemma 3.1.8 (b)), ergibt sich $s = s[t_1\sigma]_\pi \equiv_\mathcal{E} s[t_2\sigma]_\pi = t$.

Nun beweisen wir, dass $s \leftrightarrow^*_\mathcal{E} t$ ebenfalls $s \equiv_\mathcal{E} t$ impliziert. Aus $s \leftrightarrow^*_\mathcal{E} t$ folgt, dass es $s_0, \ldots, s_n$ (mit $n \geq 0$) gibt, so dass $s = s_0 \leftrightarrow_\mathcal{E} s_1 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$. Wir zeigen die Behauptung durch Induktion über n (d.h., über die Länge der Herleitung).

Im Induktionsanfang ist $n = 0$. Hier gilt $s = t$ und $s \equiv_{\mathcal{E}} t$ folgt aus der Reflexivität von $\equiv_{\mathcal{E}}$ (Lemma 3.1.10).

Im Induktionsschluss ist $n > 0$ und aus der Induktionshypothese folgt bereits $s \equiv_{\mathcal{E}} s_{n-1}$. Da $s_{n-1} \leftrightarrow_{\mathcal{E}} s_n = t$ gilt und da $\leftrightarrow_{\mathcal{E}}$ (wie oben gezeigt) in $\equiv_{\mathcal{E}}$ enthalten ist, folgt $s_{n-1} \equiv_{\mathcal{E}} s_n = t$. Aus der Transitivität von $\equiv_{\mathcal{E}}$ (Lemma 3.1.10) ergibt sich daher $s \equiv_{\mathcal{E}} t$.

Vollständigkeit:

Nun zeigen wir, dass aus $s \equiv_{\mathcal{E}} t$ auch $s \leftrightarrow^*_{\mathcal{E}} t$ folgt. Die Aussage $s \equiv_{\mathcal{E}} t$ bedeutet, dass $A \models s \equiv t$ für alle Modelle A von $\mathcal{E}$ gilt. Unser Ziel ist nun, solch ein Modell $A = (\mathcal{A}, \alpha)$ von $\mathcal{E}$ und eine Variablenbelegung β zu definieren, so dass für die Interpretation $I = (\mathcal{A}, \alpha, \beta)$ der Zusammenhang $I \models s \equiv t$ genau dann zutrifft, wenn $s \leftrightarrow^*_{\mathcal{E}} t$ gilt.

Eine erste Idee ist, als Träger $\mathcal{A}$ die Menge der Terme $\mathcal{T}(\Sigma, \mathcal{V})$ zu wählen und jeden Term als "sich selbst" zu interpretieren. Dazu müsste $\alpha_f = f$ für alle Funktionssymbole $f \in \Sigma$ sein und $\beta(x) = x$ für alle Variablen x. Durch strukturelle Induktion über den Termaufbau zeigt man leicht, dass dann $I(t) = t$ für alle Terme $t \in \mathcal{T}(\Sigma, \mathcal{V})$ ist. Das Problem hierbei wäre aber, dass zwei verschiedene Terme s und t, die unter $\leftrightarrow^*_{\mathcal{E}}$ äquivalent sind, hierbei als verschiedene Objekte des Trägers interpretiert werden würden.

Daher werden wir den Träger etwas anders wählen. Hierbei ist es wichtig, dass die Relation $\leftrightarrow^*_{\mathcal{E}}$ eine Äquivalenzrelation ist (Lemma 3.1.13). Zu jeder Äquivalenzrelation und jedem Objekt bezeichnet die *Äquivalenzklasse* des Objekts die Menge all jener Objekte, zu denen dieses Objekt äquivalent ist. In unserem Beispiel ist zu jedem Term s die Äquivalenzklasse $[s]_{\leftrightarrow^*_{\mathcal{E}}}$ definiert als $[s]_{\leftrightarrow^*_{\mathcal{E}}} = \{t \mid s \leftrightarrow^*_{\mathcal{E}} t\}$. Da $\leftrightarrow^*_{\mathcal{E}}$ eine Äquivalenzrelation ist, sind zwei Äquivalenzklassen $[s]_{\leftrightarrow^*_{\mathcal{E}}}$ und $[t]_{\leftrightarrow^*_{\mathcal{E}}}$ entweder identisch oder disjunkt. Die Menge aller Äquivalenzklassen wird als *Quotientenmenge* bezeichnet. In unserem Fall zerfällt die Menge der Terme $\mathcal{T}(\Sigma, \mathcal{V})$ also in mehrere Äquivalenzklassen und die Quotientenmenge ist $\mathcal{T}(\Sigma, \mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}} = \{\, [s]_{\leftrightarrow^*_{\mathcal{E}}} \mid s \in \mathcal{T}(\Sigma, \mathcal{V})\}$.

Unser Ziel ist nun, die Interpretation I so zu wählen, dass jeder Term t als seine *Äquivalenzklasse* interpretiert wird, d.h., wir wollen erreichen, dass $I(t) = [t]_{\leftrightarrow^*_{\mathcal{E}}}$ gilt. Dann gilt in der Tat $I \models s \equiv t$ gdw. $I(s) = I(t)$ gdw. $[s]_{\leftrightarrow^*_{\mathcal{E}}} = [t]_{\leftrightarrow^*_{\mathcal{E}}}$ gdw. $s \leftrightarrow^*_{\mathcal{E}} t$.

Der Träger $\mathcal{A}$ muss also als die Quotientenmenge $\mathcal{T}(\Sigma, \mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}}$ gewählt werden, d.h., als die Menge aller $\leftrightarrow^*_{\mathcal{E}}$-Äquivalenzklassen. Für alle Terme $f(t_1, \ldots, t_n)$ soll gelten, dass $I(f(t_1, \ldots, t_n)) = \alpha_f(I(t_1), \ldots, I(t_n)) = [f(t_1, \ldots, t_n)]_{\leftrightarrow^*_{\mathcal{E}}}$ ist. Wenn schon $I(t_i) = [t_i]_{\leftrightarrow^*_{\mathcal{E}}}$ ist, dann ergibt sich folgende Definition von α_f für alle $f \in \Sigma$ und alle $[t_i]_{\leftrightarrow^*_{\mathcal{E}}} \in \mathcal{T}(\Sigma, \mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}}$:

$$\alpha_f([t_1]_{\leftrightarrow^*_{\mathcal{E}}}, \ldots, [t_n]_{\leftrightarrow^*_{\mathcal{E}}}) = [f(t_1, \ldots, t_n)]_{\leftrightarrow^*_{\mathcal{E}}}.$$

Man beachte, dass α_f dann tatsächlich eine wohldefinierte Funktion ist, denn es ist unerheblich, welchen Term aus der Äquivalenzklasse der $[t_i]_{\leftrightarrow^*_{\mathcal{E}}}$ man verwendet, um die Äquivalenzklasse $[f(t_1, \ldots, t_n)]_{\leftrightarrow^*_{\mathcal{E}}}$ zu bilden. Wenn nämlich $t_i \leftrightarrow^*_{\mathcal{E}} s_i$ für alle i gilt, so gilt auch $f(t_1, \ldots, t_n) \leftrightarrow^*_{\mathcal{E}} f(s_1, \ldots, s_n)$, denn $\leftrightarrow^*_{\mathcal{E}}$ ist eine Kongruenzrelation (Lemma 3.1.13). Damit auch bei Variablen x jeweils $I(x) = [x]_{\leftrightarrow^*_{\mathcal{E}}}$ gilt, definieren wir

die folgende Variablenbelegung:

$$\beta(x) = [x]_{\leftrightarrow^*_{\mathcal{E}}}.$$

Für $I = (\mathcal{T}(\Sigma,\mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}}, \alpha, \beta)$ mit α und β wie oben zeigt man leicht durch strukturelle Induktion über den Aufbau der Terme, dass $I(t) = [t]_{\leftrightarrow^*_{\mathcal{E}}}$ für alle $t \in \mathcal{T}(\Sigma,\mathcal{V})$ gilt. Wie oben erläutert, gilt daher $I \models s \equiv t$ gdw. $s \leftrightarrow^*_{\mathcal{E}} t$.

Wir müssen nun noch zeigen, dass die Algebra $A = (\mathcal{T}(\Sigma,\mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}}, \alpha)$ mit α wie oben ein Modell des Gleichungssystems $\mathcal{E}$ ist. Dann folgt aus $\mathcal{E} \models s \equiv t$ nämlich $A \models s \equiv t$ und daher auch $I \models s \equiv t$, d.h., $s \leftrightarrow^*_{\mathcal{E}} t$.

Sei $u \equiv v \in \mathcal{E}$. Zu zeigen ist, dass $A \models u \equiv v$ gilt. Für jede Interpretation $J = (\mathcal{T}(\Sigma,\mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}}, \alpha, \gamma)$ (d.h., für jede Variablenbelegung γ) muss also gelten $J(u) = J(v)$. Für $x \in \mathcal{V}$ ist also $\gamma(x) \in \mathcal{T}(\Sigma,\mathcal{V})/_{\leftrightarrow^*_{\mathcal{E}}} = \{\, [s]_{\leftrightarrow^*_{\mathcal{E}}} \mid s \in \mathcal{T}(\Sigma,\mathcal{V})\}$. Für jede Variable x sei s_x ein Term aus der Äquivalenzklasse $\gamma(x)$, d.h., $\gamma(x) = [s_x]_{\leftrightarrow^*_{\mathcal{E}}}$. Weiterhin sei σ die Substitution mit $x\sigma = s_x$ für alle Variablen x, die in $\mathcal{E}$ auftreten. Dann gilt für alle Terme t in den Gleichungen $\mathcal{E}$, dass $J(t) = [t\sigma]_{\leftrightarrow^*_{\mathcal{E}}}$. (Dies lässt sich sofort durch strukturelle Induktion zeigen: Falls t eine Variable x aus $\mathcal{E}$ ist, so gilt $J(x) = \gamma(x) = [s_x]_{\leftrightarrow^*_{\mathcal{E}}} = [x\sigma]_{\leftrightarrow^*_{\mathcal{E}}}$. Falls $t = f(t_1,\ldots,t_n)$ ist (mit $n \geq 0$), so gilt $J(f(t_1,\ldots,t_n)) = \alpha_f(J(t_1),\ldots,J(t_n)) = \alpha_f([t_1\sigma]_{\leftrightarrow^*_{\mathcal{E}}},\ldots,[t_n\sigma]_{\leftrightarrow^*_{\mathcal{E}}}) = [f(t_1,\ldots,t_n)\sigma]_{\leftrightarrow^*_{\mathcal{E}}}$ nach der Induktionshypothese.)

Zu zeigen war, dass für jede Gleichung $u \equiv v$ aus $\mathcal{E}$ jeweils $J(u) = J(v)$ gilt, d.h., $[u\sigma]_{\leftrightarrow^*_{\mathcal{E}}} = [v\sigma]_{\leftrightarrow^*_{\mathcal{E}}}$ bzw. $u\sigma \leftrightarrow^*_{\mathcal{E}} v\sigma$. Dies folgt aus $u \leftrightarrow^*_{\mathcal{E}} v$ (denn $u \leftrightarrow_{\mathcal{E}} v$) und der Abgeschlossenheit von $\leftrightarrow^*_{\mathcal{E}}$ unter Substitutionen (Lemma 3.1.13 (a)). □

Damit haben wir also nun ein syntaktisches Hilfsmittel zur Verfügung, um semantische Aussagen nachzuweisen, d.h., um das Wortproblem zu lösen. In Bsp. 3.1.12 haben wir z.B. die Gleichung $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ aus den beiden Axiomen für plus (Bsp. 2.1.6) hergeleitet. Nach dem Satz von Birkhoff (dem "Korrektheits"-Teil) bedeutet dies, dass diese Aussage dann auch tatsächlich aus den Axiomen folgt (d.h. insbesondere, dass diese Aussage für das funktionale Programm, das aus diesen beiden Axiomen besteht, stimmt). Der Satz von Birkhoff sagt aber auch (im "Vollständigkeits"-Teil), dass man jede wahre Aussage auf diese Art und Weise beweisen kann. Wenn eine Gleichung aus einem Gleichungssystem folgt, so gibt es also auch eine Herleitung, mit der man sie beweisen kann. Mit diesem Verfahren wollen wir nun zeigen, dass in der Tat in allen Gruppen die inverse Funktion selbstinvers ist.

Beispiel 3.1.15 Sei $\mathcal{E} = \{(2.1), (2.2), (2.3)\}$ das Gleichungssystem der Gruppenaxiome aus Bsp. 2.1.7. Wir wollen zeigen, dass $\mathsf{i}(\mathsf{i}(n)) \equiv_{\mathcal{E}} n$ für alle Elemente n gilt. Nach dem Satz von Birkhoff genügt es hierfür, $\mathsf{i}(\mathsf{i}(n)) \leftrightarrow^*_{\mathcal{E}} n$ zu zeigen und falls dieser Satz tatsächlich wahr ist, muss es auch eine entsprechende solche Herleitung geben. Die Stellen, an denen Teilterme ersetzt werden, sind wieder durch Unterstreichung markiert. Gleichungen werden "von links nach rechts" angewendet und werden deshalb wieder je nach Bedarf als $t_1 \equiv t_2$ oder $t_2 \equiv t_1$ geschrieben.

$\underline{\mathsf{i}(\mathsf{i}(n))}$	$x \equiv \mathsf{f}(x, \mathsf{e})$	$\sigma = \{x/\mathsf{i}(\mathsf{i}(n))\}$
$\mathsf{f}(\mathsf{i}^2(n), \underline{\mathsf{e}})$	$\mathsf{e} \equiv \mathsf{f}(x, \mathsf{i}(x))$	$\sigma = \{x/\mathsf{i}^3(n)\}$
$\underline{\mathsf{f}(\mathsf{i}^2(n), \mathsf{f}(\mathsf{i}^3(n), \mathsf{i}^4(n)))}$	$\mathsf{f}(x, \mathsf{f}(y, z)) \equiv \mathsf{f}(\mathsf{f}(x, y), z)$	$\sigma = \{x/\mathsf{i}^2(n), y/\mathsf{i}^3(n), z/\mathsf{i}^4(n)\}$
$\mathsf{f}(\underline{\mathsf{f}(\mathsf{i}^2(n), \mathsf{i}^3(n))}, \mathsf{i}^4(n))$	$\mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e}$	$\sigma = \{x/\mathsf{i}^2(n)\}$
$\mathsf{f}(\underline{\mathsf{e}}, \mathsf{i}^4(n))$	$x \equiv \mathsf{f}(x, \mathsf{e})$	$\sigma = \{x/\mathsf{e}\}$
$\underline{\mathsf{f}(\mathsf{f}(\mathsf{e}, \mathsf{e}), \mathsf{i}^4(n))}$	$\mathsf{f}(\mathsf{f}(x, y), z) \equiv \mathsf{f}(x, \mathsf{f}(y, z))$	$\sigma = \{x/\mathsf{e}, y/\mathsf{e}, z/\mathsf{i}^4(n)\}$
$\mathsf{f}(\mathsf{e}, \mathsf{f}(\underline{\mathsf{e}}, \mathsf{i}^4(n)))$	$\mathsf{e} \equiv \mathsf{f}(x, \mathsf{i}(x))$	$\sigma = \{x/\mathsf{i}^2(n)\}$
$\mathsf{f}(\mathsf{e}, \underline{\mathsf{f}(\mathsf{f}(\mathsf{i}^2(n), \mathsf{i}^3(n)), \mathsf{i}^4(n))})$	$\mathsf{f}(\mathsf{f}(x, y), z) \equiv \mathsf{f}(x, \mathsf{f}(y, z))$	$\sigma = \{x/\mathsf{i}^2(n), y/\mathsf{i}^3(n), z/\mathsf{i}^4(n)\}$
$\underline{\mathsf{f}(\mathsf{e}, \mathsf{f}(\mathsf{i}^2(n), \mathsf{f}(\mathsf{i}^3(n), \mathsf{i}^4(n))))}$	$\mathsf{f}(x, \mathsf{f}(y, z)) \equiv \mathsf{f}(\mathsf{f}(x, y), z)$	$\sigma = \{x/\mathsf{e}, y/\mathsf{i}^2(n), z/\mathsf{f}(\mathsf{i}^3(n), \mathsf{i}^4(n))\}$
$\underline{\mathsf{f}(\mathsf{f}(\mathsf{e}, \mathsf{i}^2(n)), \mathsf{f}(\mathsf{i}^3(n), \mathsf{i}^4(n)))}$	$\mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e}$	$\sigma = \{x/\mathsf{i}^3(n)\}$
$\underline{\mathsf{f}(\mathsf{f}(\mathsf{e}, \mathsf{i}^2(n)), \mathsf{e})}$	$\mathsf{f}(x, \mathsf{e}) \equiv x$	$\sigma = \{x/\mathsf{f}(\mathsf{e}, \mathsf{i}^2(n))\}$
$\mathsf{f}(\underline{\mathsf{e}}, \mathsf{i}^2(n))$	$\mathsf{e} \equiv \mathsf{f}(x, \mathsf{i}(x))$	$\sigma = \{x/n\}$
$\underline{\mathsf{f}(\mathsf{f}(n, \mathsf{i}(n)), \mathsf{i}^2(n))}$	$\mathsf{f}(\mathsf{f}(x, y), z) \equiv \mathsf{f}(x, \mathsf{f}(y, z))$	$\sigma = \{x/n, y/\mathsf{i}(n), z/\mathsf{i}^2(n)\}$
$\mathsf{f}(n, \underline{\mathsf{f}(\mathsf{i}(n), \mathsf{i}^2(n))})$	$\mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e}$	$\sigma = \{x/\mathsf{i}(n)\}$
$\underline{\mathsf{f}(n, \mathsf{e})}$	$\mathsf{f}(x, \mathsf{e}) \equiv x$	$\sigma = \{x/n\}$
n		

Aufgrund von Satz 3.1.14 ist man mit der Beweisrelation $\leftrightarrow_{\mathcal{E}}^*$ in der Lage, das Wortproblem $s \equiv_{\mathcal{E}} t$ ohne Rückgriff auf semantische Begriffe zu beweisen. Von Vorteil ist dabei insbesondere, dass der Nachweis von $\mathcal{E} \models s \equiv t$ *systematisch* geführt werden kann. Anders gesagt, man kann nach einer Herleitung von $s \equiv t$ so suchen, dass man immer erfolgreich ist, wenn $\mathcal{E} \models s \equiv t$ tatsächlich gilt. Dies gilt, da $\leftrightarrow_{\mathcal{E}}^*$ *semi-entscheidbar* (oder *rekursiv aufzählbar*) ist. Es gibt also ein Verfahren, das bei der Eingabe von s und t mit Erfolg terminiert, wenn $s \leftrightarrow_{\mathcal{E}}^* t$ gilt. Wenn $s \leftrightarrow_{\mathcal{E}}^* t$ hingegen nicht gilt, so kann es sein, dass das Verfahren nicht terminiert.

Das Verfahren arbeitet wie folgt bei der Eingabe zweier Terme s und t. Falls $s = t$ gilt, so ist das Problem trivialerweise gelöst. Anderenfalls erzeugt man einen *Suchbaum* folgender Art: Der Wurzelknoten n_0 wird mit s markiert. Für jeden Term u mit $s \leftrightarrow_{\mathcal{E}} u$ hat n_0 einen direkten Nachfolgerknoten n_i, der mit u markiert ist. Genauso erhält man alle Nachfolgerknoten von n_i, etc. Wenn die *Variablenbedingung* $\mathcal{V}(t_1) = \mathcal{V}(t_2)$ für alle Gleichungen $t_1 \equiv t_2 \in \mathcal{E}$ gilt, so hat jeder Knoten nur endlich viele Nachfolger. (Denn es gibt in jedem Term nur endlich viele Teilterme und endlich viele Gleichungen, die darauf passen können. Das Verhalten des Matchers auf den Variablen der Gleichung ist eindeutig festgelegt, wenn $\mathcal{V}(t_1) = \mathcal{V}(t_2)$ gilt, siehe unten.)

Wenn $u_1, \ldots, u_n$ die Folge der Knotenmarkierungen eines Pfades von der Wurzel zu einem Knoten ist, so gilt $u_1 \leftrightarrow_{\mathcal{E}} u_2 \leftrightarrow_{\mathcal{E}} \ldots \leftrightarrow_{\mathcal{E}} u_n$, d.h., $u_1 \leftrightarrow_{\mathcal{E}}^* u_n$. Jeder Pfad des Suchbaums repräsentiert also eine Herleitung und jede Herleitung wird als Pfad des Suchbaums nach endlich vielen Schritten erzeugt. Das Verfahren endet mit Erfolg, falls ein Knoten mit Markierung t gebildet wurde. Für $\mathcal{E} \models s \equiv t$ terminiert das Verfahren, für $\mathcal{E} \not\models s \equiv t$ dagegen nicht.

Für endliches $\mathcal{E}$ hat jeder Suchbaum endlichen Verzweigungsgrad, denn für jeden Term s gibt es maximal $|Occ(s)| \times 2 \times |\mathcal{E}|$ Terme t mit $s \leftrightarrow_{\mathcal{E}} t$. Dies liegt an der geforderten Variablenbedingung $\mathcal{V}(t_1) = \mathcal{V}(t_2)$ für alle Gleichungen $t_1 \equiv t_2 \in \mathcal{E}$. Mit $s \leftrightarrow_{\mathcal{E}} t$ gilt $t_1\sigma = s|_\pi$ und $t = s[t_2\sigma]_\pi$ für eine Gleichung $t_1 \equiv t_2$ oder $t_2 \equiv t_1$ aus $\mathcal{E}$ und damit ist σ

hinreichend eindeutig festgelegt. Für jeden weiteren Matcher θ mit $t_1\theta = s|_\pi$ gilt nämlich $t_2\theta = t_2\sigma$ und somit $s[t_2\theta]_\pi = s[t_2\sigma]_\pi = t$.

Gibt man jedoch die obige Variablenbedingung auf, so kann der Verzweigungsgrad unendlich sein. Im Gruppen-Beispiel erfüllt die Gleichung $\mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e}$ die Variablenbedingung nicht. Daher gibt es unendlich viele Terme t mit $\mathsf{e} \leftrightarrow_\mathcal{E} t$, denn es gibt unendlich viele Matcher $\sigma = \{x/q\}$ mit $\mathsf{e}\sigma = \mathsf{e}$. Für jeden Term $\mathsf{f}(q, \mathsf{i}(q))$ mit beliebigem Term q gilt daher $\mathsf{e} \leftrightarrow_\mathcal{E} \mathsf{f}(q, \mathsf{i}(q))$. Man muss also im allgemeinen bei der Suche einer Herleitung geeignete Matcher "raten". Wie man sich leicht anhand von Bsp. 3.1.15 überzeugt, ist gerade dies der schwierige Schritt bei der Suche nach einer Herleitung.

Da $\leftrightarrow^*_\mathcal{E}$ aber semi-entscheidbar ist, lässt sich das Suchbaumverfahren zur Behandlung dieser Situationen so modifizieren, dass Semi-Vollständigkeit (und Korrektheit) erhalten bleiben. Durch den nicht-endlichen Verzweigungsgrad wird es dadurch jedoch für eine praktische Anwendung unbrauchbar.

Aber auch für Gleichungssysteme, deren Gleichungen der Variablenbedingung genügen, ist das naive Suchbaumverfahren in der Praxis ungeeignet. Dies liegt daran, dass mit einem Verzweigungsgrad > 1 beim Aufbau des Suchbaums die Anzahl der Knoten exponentiell mit der Tiefe des Baums wächst.

Hinter dem Verzweigungsgrad des Suchbaums steckt der Indeterminismus der Ersetzungsrelation. Für einen Knoten n_i mit Markierung u_i müssen zunächst alle Stellen $\pi \in Occ(u_i)$ betrachtet werden. Für jeden Teilterm $u_i|_\pi$ von u_i muss dann für jede Gleichung $t_1 \equiv t_2$ oder $t_2 \equiv t_1$ in $\mathcal{E}$, für die t_1 den Teilterm $u_i|_\pi$ matcht, ein Nachfolger von n_i gebildet werden.

Allgemein ist zum Führen eines mathematischen Beweises eine gewisse "Intelligenz", Wissen und "Intuition" erforderlich. In unserem Kontext heißt das, dass ein Experte zielgerichtet die "richtige" Stelle π und die "richtige" Gleichung $t_1 \equiv t_2$ auswählt, um die gewünschte Herleitung zu bilden. Das naive Suchbaumverfahren "umgeht" dagegen das Problem, indem alle Konsequenzen gebildet werden. Der Preis, der dafür gezahlt wird, besteht in dem hohen (und letztendlich nicht tolerierbaren) Aufwand des Suchbaumverfahrens.

3.2 Der Kongruenzabschluss bei Grundidentitäten

Wir wollen nun auf dem Resultat des vorigen Abschnitts (dem Satz von Birkhoff, Satz 3.1.14) aufbauen und bessere Techniken entwickeln, um das Wortproblem zu lösen. Das Wortproblem, (d.h. die Relation $\equiv_\mathcal{E}$) ist im allgemeinen nicht entscheidbar. Es gibt also für beliebige Gleichungssysteme $\mathcal{E}$ kein automatisches Verfahren, das bei Eingabe von zwei Termen s und t immer terminiert und herausfindet, ob $s \equiv_\mathcal{E} t$ gilt. Allerdings existiert eine wichtige Teilklasse von Gleichungssystemen $\mathcal{E}$, bei der das Wortproblem tatsächlich entscheidbar ist, nämlich Gleichungen *ohne Variablen*. In diesem Abschnitt soll dieses Resultat und das automatische Verfahren des *Kongruenzabschlusses* vorgestellt werden, das die Gültigkeit solcher Gleichungen nachweisen kann. Das hier vorgestellte Verfahren beruht auf [Sho78]; andere Verfahren hierzu findet man u.a. in [DST80, NO80].

Dieses (und darauf aufbauende) Verfahren haben große Anwendungsrelevanz unter anderem in folgenden Gebieten:

- *Compilerbau*
  Compiler führen üblicherweise mehrere Optimierungen durch, um die Effizienz des compilierten Codes zu verbessern. Eine der grundlegenden Optimierungstechniken ist das *common subexpression problem* [DST80], bei dem versucht wird, mehrfach auftretende Teilausdrücke (d.h., Teilterme) zu erkennen und nur einmal auszuwerten. Hierbei sind zwei Teilausdrücke nicht nur dann "gleich", wenn sie syntaktisch identisch sind, sondern auch dann, wenn dies aus den durch den vorangegangenen Code gegebenen Gleichungen folgt.

- *Programmverifikation*
  Ähnliche Fragestellungen treten natürlich auch bei der Programmverifikation und beim automatischen Beweisen auf. Aus dem Kongruenzabschluss-Verfahren kann man ein Entscheidungsverfahren für die Allgemeingültigkeit universeller prädikatenlogischer Formeln erhalten [NO80]. Darüber hinaus lässt sich auf diese Weise auch ein Entscheidungsverfahren für universelle Aussagen über rekursiv definierte Datenstrukturen (wie Listen, Bäume, natürliche Zahlen, etc.) gewinnen [Opp80, NO80].

- *Kombination von Entscheidungsverfahren*
  Eines der zur Zeit aktuellsten Forschungsbereiche (insbesondere in der automatisierten Programmverifikation) ist die Frage, wie man Entscheidungsverfahren für verschiedene Theorien kombiniert. Hierzu werden Verfahren verwendet, die ebenfalls auf dem Kongruenzabschluss basieren [NO79, Sho84, Zha92, RS02].

Definition 3.2.1 (Grundidentität) *Eine Termgleichung $s \equiv t$ ist eine* Grundidentität*, falls sie keine Variablen enthält, d.h., falls $\mathcal{V}(s) = \mathcal{V}(t) = \varnothing$.*

Beispiel 3.2.2 Betrachten wir ein imperatives Programm, bei dem `i`, `j`, `k`, `l`, `m` Variablen (für natürliche Zahlen) und `f` und `g` Variablen für Arrays sind. Das Programm habe nun folgenden Programmtext:

```
 ...
i = j;
k = l;
f[i] = g[k];
if (j == f[j]) {

    m = g[l];
     ...          (*)

 }
  ...
```

Die Frage ist nun, ob an der Stelle `(*)` der Zusammenhang `f[m] = g[k]` gilt. Dies kann zum einen für die Verifikation des Programms interessant sein und zum anderen für den Compiler, der dann weiß, dass er beim Vorkommen der beiden Teilausdrücke `f[m]` und `g[k]` nur einen davon auswerten muss, da der andere denselben Wert hat.

Anders ausgedrückt bedeutet dies, ob $\mathsf{f}(\mathsf{m}) \equiv_{\mathcal{E}} \mathsf{g}(\mathsf{k})$ gilt, wobei

$$\mathcal{E} = \{\mathsf{i} \equiv \mathsf{j}, \mathsf{k} \equiv \mathsf{l}, \mathsf{f}(\mathsf{i}) \equiv \mathsf{g}(\mathsf{k}), \mathsf{j} \equiv \mathsf{f}(\mathsf{j}), \mathsf{m} \equiv \mathsf{g}(\mathsf{l})\}$$

ist. (Um die Gleichungen im allgemeinen bestimmen zu können, muss man genau analysieren, welche Werte durch Seiteneffekte etc. verändert wurden.) Hierbei sind `i`, `j`, `k`, `l` und `m` nun Konstanten und `f` und `g` einstellige Funktionssymbole. (Die Variablen des Programms entsprechen *Konstanten*, da die Frage ist, ob `f[m] = g[k]` für *genau diese* Werte von `m` und `k` gilt und nicht, ob `f[m] = g[k]` für *alle* Werte von `m` und `k` gilt.) Hierdurch wird deutlich, dass die Frage des Wortproblems über Grundidentitäten durchaus anwendungsrelevant ist.

Nach dem Satz von Birkhoff (Satz 3.1.14) wissen wir, dass die semantische Relation $\equiv_{\mathcal{E}}$ und die syntaktische Relation $\leftrightarrow_{\mathcal{E}}^*$ identisch sind. Um $\mathsf{f}(\mathsf{m}) \equiv_{\mathcal{E}} \mathsf{g}(\mathsf{k})$ zu beweisen, können wir also eine Herleitung mit Hilfe von $\leftrightarrow_{\mathcal{E}}^*$ angeben. Beispielsweise kann man die Aussage wie folgt beweisen:

$$\begin{array}{lll}
 & \mathsf{f}(\mathsf{m}) & \\
\leftrightarrow_{\mathcal{E}} & \mathsf{f}(\mathsf{g}(\mathsf{l})) & \text{mit } \mathsf{m} \equiv \mathsf{g}(\mathsf{l}) \\
\leftrightarrow_{\mathcal{E}} & \mathsf{f}(\mathsf{g}(\mathsf{k})) & \text{mit } \mathsf{k} \equiv \mathsf{l} \\
\leftrightarrow_{\mathcal{E}} & \mathsf{f}(\mathsf{f}(\mathsf{i})) & \text{mit } \mathsf{f}(\mathsf{i}) \equiv \mathsf{g}(\mathsf{k}) \\
\leftrightarrow_{\mathcal{E}} & \mathsf{f}(\mathsf{f}(\mathsf{j})) & \text{mit } \mathsf{i} \equiv \mathsf{j} \\
\leftrightarrow_{\mathcal{E}} & \mathsf{f}(\mathsf{j}) & \text{mit } \mathsf{j} \equiv \mathsf{f}(\mathsf{j}) \\
\leftrightarrow_{\mathcal{E}} & \mathsf{f}(\mathsf{i}) & \text{mit } \mathsf{i} \equiv \mathsf{j} \\
\leftrightarrow_{\mathcal{E}} & \mathsf{g}(\mathsf{k}) & \text{mit } \mathsf{f}(\mathsf{i}) \equiv \mathsf{g}(\mathsf{k})
\end{array}$$

Der Nachteil ist, dass wir bislang kein systematisches Verfahren haben, um die richtige Herleitung automatisch zu finden. Insofern ist die Relation $\leftrightarrow_{\mathcal{E}}^*$ für automatische Herleitungen ungeeignet.

In diesem Abschnitt wollen wir ein Verfahren vorstellen, dass das Wortproblem über Grundidentitäten automatisch löst. Unser Ziel ist also, die Menge aller Gleichungen $s \equiv t$ zu charakterisieren, die aus einer Menge von Grundidentitäten $\mathcal{E}$ folgen. Hierbei müssen wir keine Substitutionen mehr betrachten, da wir nur noch Grundterme untersuchen. Aus einer Menge von Grundidentitäten $\mathcal{E}$ folgen dann nur die Gleichungen, die sich aufgrund der Reflexivität, Symmetrie, Transitivität und Kongruenz aus $\mathcal{E}$ ergeben. Dies führt zu den folgenden Mengen, die sich durch *direkte* einmalige Anwendung dieser Gesetzmäßigkeiten aus $\mathcal{E}$ ergeben.

Definition 3.2.3 (Direkte Anwendung von Reflexivität, Symmetrie, etc.)
Für jede Menge $\mathcal{E}$ von Grundidentitäten über der Signatur Σ definieren wir

- $R = \{t \equiv t \mid t \in \mathcal{T}(\Sigma)\}$
- $S(\mathcal{E}) = \{t \equiv s \mid s \equiv t \ \in \mathcal{E}\}$
- $T(\mathcal{E}) = \{s \equiv r \mid$ *es existiert ein* $t \in \mathcal{T}(\Sigma)$ *mit* $s \equiv t \ \in \mathcal{E}$ *und* $t \equiv r \ \in \mathcal{E}\}$
- $C(\mathcal{E}) = \{f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n) \mid f \in \Sigma, s_i \equiv t_i \ \in \mathcal{E}$ *für alle* $1 \leq i \leq n\}$

Beispiel 3.2.4 Für $\mathcal{E}$ aus Bsp. 3.2.2 und $\Sigma = \{\mathsf{i}, \mathsf{j}, \mathsf{k}, \mathsf{l}, \mathsf{m}, \mathsf{f}, \mathsf{g}\}$ erhält man

- $R = \{\mathsf{i} \equiv \mathsf{i}, \mathsf{j} \equiv \mathsf{j}, \mathsf{f}(\mathsf{i}) \equiv \mathsf{f}(\mathsf{i}), \ldots\}$
- $S(\mathcal{E}) = \{\mathsf{j} \equiv \mathsf{i}, \mathsf{l} \equiv \mathsf{k}, \mathsf{g}(\mathsf{k}) \equiv \mathsf{f}(\mathsf{i}), \mathsf{f}(\mathsf{j}) \equiv \mathsf{j}, \mathsf{g}(\mathsf{l}) \equiv \mathsf{m}\}$
- $T(\mathcal{E}) = \{\mathsf{i} \equiv \mathsf{f}(\mathsf{j})\}$
- $C(\mathcal{E}) = \{\mathsf{f}(\mathsf{i}) \equiv \mathsf{f}(\mathsf{j}), \mathsf{g}(\mathsf{i}) \equiv \mathsf{g}(\mathsf{j}), \ldots\}$

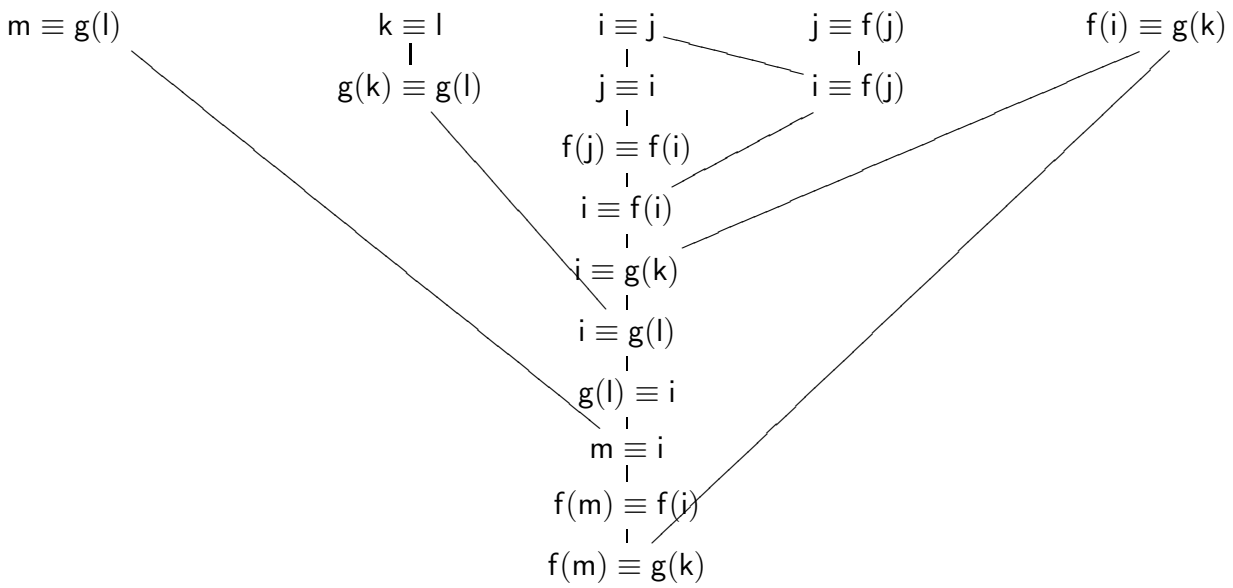
Man erkennt, dass für eine Menge von Grundidentitäten $\mathcal{E}$ die Menge $\mathcal{E} \cup R \cup S(\mathcal{E}) \cup T(\mathcal{E}) \cup C(\mathcal{E})$ zwar lauter Gleichheiten enthält, die aus $\mathcal{E}$ folgen, aber sie enthält noch nicht *alle* diese Gleichheiten. Beispielsweise kann es ja nötig sein, dass man bei der Anwendung der Kongruenz auf Gleichheiten Bezug nimmt, die nicht aus $\mathcal{E}$ direkt stammen, sondern aus $R \cup S(\mathcal{E}) \cup T(\mathcal{E}) \cup C(\mathcal{E})$. Mit anderen Worten, man muss die Anwendung der Symmetrie, Transitivität und Kongruenz *wiederholen* und zwar (potentiell) unendlich oft. Dies führt zu dem Begriff des *Kongruenzabschlusses* (engl. congruence closure).

Definition 3.2.5 (Kongruenzabschluss) *Für jede Menge $\mathcal{E}$ von Grundidentitäten über der Signatur Σ definieren wir die Mengen $\mathcal{E}_0, \mathcal{E}_1, \ldots$ wie folgt:*

- $\mathcal{E}_0 = \mathcal{E} \cup R$
- $\mathcal{E}_{i+1} = \mathcal{E}_i \cup S(\mathcal{E}_i) \cup T(\mathcal{E}_i) \cup C(\mathcal{E}_i)$

Der Kongruenzabschluss $CC(\mathcal{E})$ *von $\mathcal{E}$ ist definiert als der "Limes" der Folge $\mathcal{E}_0, \mathcal{E}_1, \ldots$, d.h., $CC(\mathcal{E}) = \bigcup_{i \in \mathbb{N}} \mathcal{E}_i$.*

Beispiel 3.2.6 Es gilt $\mathsf{f}(\mathsf{m}) \equiv \mathsf{g}(\mathsf{k}) \in \mathcal{E}_9 \subseteq CC(\mathcal{E})$. Im folgenden Baum sind die Gleichungen in der obersten Zeile in $\mathcal{E}_0$ enthalten, die in der nächsten Zeile sind in $\mathcal{E}_1$, etc. Die Striche geben jeweils an, aus welchen bisherigen Gleichungen sich die jeweilige nächste Gleichung ergibt.



Man kann zeigen, dass der Kongruenzabschluss von $\mathcal{E}$ die kleinste Kongruenzrelation beschreibt, in der alle Gleichungen aus $\mathcal{E}$ gelten. Es ist offensichtlich, dass der Kongruenzabschluss nur Gleichungen enthält, die aus $\mathcal{E}$ folgen. Durch die obige Herleitung wird also wie gewünscht $\mathsf{f}(\mathsf{m}) \equiv_{\mathcal{E}} \mathsf{g}(\mathsf{k})$ bewiesen. In der Tat gilt aber auch der Umkehrschluss, d.h., auf diese Weise erhält man alle Gleichungen, die aus $\mathcal{E}$ folgen.

Satz 3.2.7 (Kongruenzabschluss ist korrekt und vollständig) *Sei $\mathcal{E}$ eine Menge von Grundidentitäten über Σ und sei $s, t \in \mathcal{T}(\Sigma)$. Dann gilt $s \equiv_{\mathcal{E}} t$ gdw. $s \equiv t \ \in CC(\mathcal{E})$.*

Beweis.

Korrektheit:

Wir zeigen, dass für alle $i \in \mathbb{N}$ aus $s \equiv t \in \mathcal{E}_i$ die Aussage $s \equiv_{\mathcal{E}} t$ folgt. Dann ergibt sich die Behauptung sofort, da $CC(\mathcal{E}) = \bigcup_{i \in \mathbb{N}} \mathcal{E}_i$.

Hierzu verwenden wir Induktion über i. Im Induktionsanfang ist $i = 0$ und $\mathcal{E}_0 = \mathcal{E} \cup R$. Somit ist hier die Aussage trivial.

Im Induktionsschluss sei $s \equiv t \in \mathcal{E}_{i+1} = \mathcal{E}_i \cup S(\mathcal{E}_i) \cup T(\mathcal{E}_i) \cup C(\mathcal{E}_i)$. Nach der Induktionshypothese folgt für alle Gleichungen $s' \equiv t' \ \in \mathcal{E}_i$, dass $s' \equiv_{\mathcal{E}} t'$ gilt. Damit ergibt sich dies auch für alle $s \equiv t \in \mathcal{E}_{i+1}$, da $\equiv_{\mathcal{E}}$ eine Kongruenzrelation ist (Lemma 3.1.10).

Vollständigkeit:

Wir verzichten hier auf den Beweis der Vollständigkeit, da wir in Satz 3.2.12 eine stärkere Aussage beweisen, aus der die Vollständigkeit von $CC(\mathcal{E})$ sofort folgt. □

Eine erste Idee für ein Verfahren zur Lösung des Wortproblems bei Grundidentitäten wäre also, sukzessive die Mengen $\mathcal{E}_0, \mathcal{E}_1, \ldots$ zu erzeugen, bis man eine Menge $\mathcal{E}_i$ gefunden hat, die die gesuchte Gleichung $s \equiv t$ enthält. Der Kongruenzabschluss $CC(\mathcal{E})$ ist aber im allgemeinen unendlich, d.h., man hat im allgemeinen $\mathcal{E}_{i+1} \neq \mathcal{E}_i$. Beispielsweise gilt für $\mathcal{E}$ aus Bsp. 3.2.2, dass $\mathsf{i} \equiv \mathsf{j} \ \in \mathcal{E}_0$, $\mathsf{g}(\mathsf{i}) \equiv \mathsf{g}(\mathsf{j}) \ \in \mathcal{E}_1 \setminus \mathcal{E}_0$, $\mathsf{g}(\mathsf{g}(\mathsf{i})) \equiv \mathsf{g}(\mathsf{g}(\mathsf{j})) \ \in \mathcal{E}_2 \setminus \mathcal{E}_1$, $\ldots$, $\mathsf{g}^n(\mathsf{i}) \equiv \mathsf{g}^n(\mathsf{j}) \ \in \mathcal{E}_n \setminus \mathcal{E}_{n-1}$, etc. Wenn die Gleichung $s \equiv t$ daher nicht aus $\mathcal{E}$ folgt, dann terminiert dieses Verfahren nicht. Es handelt sich also wieder bloß um ein Semi-Entscheidungsverfahren.

Im Unterschied zum allgemeinen Wortproblem ist das Wortproblem bei Grundidentitäten aber entscheidbar. Um z.B. festzustellen, dass im obigen Beispiel die Gleichung $\mathsf{i} \equiv \mathsf{k}$ *nicht* aus $\mathcal{E}$ folgt, stellt sich die Frage, ob man wirklich alle (unendlich vielen) Iterationen $\mathcal{E}_0, \mathcal{E}_1, \ldots$ betrachten muss, oder ob es reicht, dafür nur endlich viele zu untersuchen. Es stellt sich heraus, dass es bei Grundidentitäten $\mathcal{E}$ reicht, nur einen endlichen Suchraum zu durchlaufen: Um herauszufinden, ob $s \equiv_{\mathcal{E}} t$ gilt, muss man nur die (endlich vielen) Terme betrachten, die in der Menge $\mathcal{E}$ oder in den Eingabetermen s und t auftreten.

Definition 3.2.8 (Teiltermmenge) *Zu jedem Term s sei $Subterms(s) = \{s|_\pi \mid \pi \in Occ(s)\}$ die Menge seiner Teilterme. Zu einer Menge von Gleichungen $\mathcal{E}$ definieren wir*

$$Subterms(\mathcal{E}) \ = \bigcup_{s \equiv t \ \in \mathcal{E}} Subterms(s) \cup Subterms(t).$$

Für jeden Term s ist die Menge $Subterms(s)$ endlich, denn jeder Term hat nur endlich viele Teilterme. Wenn das Gleichungssystem $\mathcal{E}$ endlich ist, so ist damit auch $Subterms(\mathcal{E})$ endlich. Es zeigt sich, dass wir bei der Bildung des Kongruenzabschlusses nicht *alle* möglichen Terme betrachten müssen, sondern nur die Terme aus $S = Subterms(\mathcal{E}) \cup Subterms(s) \cup Subterms(t)$, wenn wir untersuchen wollen, ob $s \equiv_{\mathcal{E}} t$ gilt. Dies führt zur Definition des Kongruenzabschlusses *bezüglich einer Menge von Termen* S.

Definition 3.2.9 (Kongruenzabschluss bezüglich einer Menge von Termen) *Sei $\mathcal{E}$ eine Menge von Grundidentitäten über der Signatur Σ und seien $s, t \in \mathcal{T}(\Sigma)$. Weiter sei $S = Subterms(\mathcal{E}) \cup Subterms(s) \cup Subterms(t)$. Dann definieren wir die Mengen $\mathcal{E}_0^S, \mathcal{E}_1^S, \ldots$ wie folgt:*

- $\mathcal{E}_0^S = (\mathcal{E} \cup R) \cap S \times S$
- $\mathcal{E}_{i+1}^S = (\mathcal{E}_i^S \cup S(\mathcal{E}_i^S) \cup T(\mathcal{E}_i^S) \cup C(\mathcal{E}_i^S)) \cap S \times S$.

Der Kongruenzabschluss $CC^S(\mathcal{E})$ *von* $\mathcal{E}$ bezüglich S *ist definiert als der "Limes" der Folge $\mathcal{E}_0^S, \mathcal{E}_1^S, \ldots$, d.h., $CC^S(\mathcal{E}) = \bigcup_{i \in \mathbb{N}} \mathcal{E}_i^S$.*

In der obigen Definition von $\mathcal{E}_{i+1}^S$ betrachten wir Gleichungen der Einfachheit halber als Paare von Termen. Alternativ könnte man z.B. auch schreiben

$$\mathcal{E}_{i+1}^S = (\mathcal{E}_i^S \cup S(\mathcal{E}_i^S) \cup T(\mathcal{E}_i^S) \cup C(\mathcal{E}_i^S)) \cap \{u \equiv v \mid u, v \in S\}.$$

Beispiel 3.2.10 Für $\mathcal{E}$ aus Bsp. 3.2.2 und $s = \mathsf{f(m)}$, $t = \mathsf{g(k)}$ erhält man $S = \{\mathsf{i}, \mathsf{j}, \mathsf{k}, \mathsf{l}, \mathsf{m}, \mathsf{f(i)}, \mathsf{f(j)}, \mathsf{f(m)}, \mathsf{g(k)}, \mathsf{g(l)}\}$. Im Unterschied zu $\mathcal{E}_1$ enthält $\mathcal{E}_1^S$ also nur Gleichungen zwischen Termen aus S. Eine Gleichung wie $\mathsf{g(i)} \equiv \mathsf{g(j)}$ fehlt also. Genauer haben wir

$$\begin{aligned} \mathcal{E}_1^S \;=\; & \mathcal{E} \cup \\ & \{t \equiv t \mid t \in S\} \cup \\ & \{\mathsf{j} \equiv \mathsf{i}, \mathsf{l} \equiv \mathsf{k}, \mathsf{g(k)} \equiv \mathsf{f(i)}, \mathsf{f(j)} \equiv \mathsf{j}, \mathsf{g(l)} \equiv \mathsf{m}\} \cup \\ & \{\mathsf{i} \equiv \mathsf{f(j)}\} \cup \\ & \{\mathsf{f(i)} \equiv \mathsf{f(j)}, \mathsf{g(k)} \equiv \mathsf{g(l)}\} \end{aligned}$$

Da S endlich ist, existieren nur endliche viele Gleichungen $s \equiv t$ mit $s, t \in S$. Alle $\mathcal{E}_i^S$ sind daher endlich und es existiert ein $i \in \mathbb{N}$ mit $\mathcal{E}_i = \mathcal{E}_{i+1}$. Der Kongruenzabschluss bzgl. S wird also *nach endlich vielen Schritten* erreicht.

Lemma 3.2.11 ($CC^S(\mathcal{E})$ wird nach endlich vielen Schritten erreicht) *Sei $\mathcal{E}$ eine endliche Menge von Grundidentitäten und $S = Subterms(\mathcal{E}) \cup Subterms(s) \cup Subterms(t)$ für $s, t \in \mathcal{T}(\Sigma)$. Dann existiert ein $i \in \mathbb{N}$, so dass $\mathcal{E}_i^S = \mathcal{E}_{i+1}^S$. Daraus folgt $CC^S(\mathcal{E}) = \mathcal{E}_i^S$.*

Beweis. Es gilt $\mathcal{E}_i^S \subseteq \mathcal{E}_{i+1}^S$ für alle $i \in \mathbb{N}$. Da $\mathcal{E}_i^S \subseteq S \times S$ für alle $i \in \mathbb{N}$ und da $S \times S$ endlich ist, muss es ein $i \in \mathbb{N}$ geben mit $\mathcal{E}_i^S = \mathcal{E}_{i+1}^S$. Daraus folgt dann $\mathcal{E}_i^S = \mathcal{E}_j^S$ für alle $j \geq i$ und somit $\mathcal{E}_i^S = \bigcup_{j \in \mathbb{N}} \mathcal{E}_j^S = CC^S(\mathcal{E})$. □

Der folgende Satz zeigt, dass dieser eingeschränkte (und endliche) Kongruenzabschluss bereits ebenfalls korrekt und vollständig ist. Mit anderen Worten, um $s \equiv_{\mathcal{E}} t$ zu untersuchen,

muss man nur $CC^S(\mathcal{E})$ berechnen. Diese Menge ist endlich und lässt sich mit endlich vielen Iterationen berechnen. Dann gilt $s \equiv_\mathcal{E} t$ genau dann, wenn die Gleichung $s \equiv t$ in $CC^S(\mathcal{E})$ enthalten ist.

Satz 3.2.12 (Kongruenzabschluss bzgl. S ist korrekt und vollständig) *Sei $\mathcal{E}$ eine Menge von Grundidentitäten über der Signatur Σ und seien $s,t \in \mathcal{T}(\Sigma)$. Weiter sei $Subterms(\mathcal{E}) \cup Subterms(s) \cup Subterms(t) \subseteq S$. Dann gilt $s \equiv_\mathcal{E} t$ gdw. $s \equiv t \in CC^S(\mathcal{E})$.*

Beweis.

Korrektheit:

Offensichtlich ist $CC^S(\mathcal{E}) \subseteq CC(\mathcal{E})$. Somit folgt aus $s \equiv t \in CC^S(\mathcal{E})$ sofort $s \equiv_\mathcal{E} t$ aufgrund der Korrektheit des Kongruenzabschlusses (Satz 3.2.7).

Vollständigkeit:

Nach dem Satz von Birkhoff (Satz 3.1.14) ist $s \equiv_\mathcal{E} t$ gleichbedeutend zu $s \leftrightarrow^*_\mathcal{E} t$. Wir zeigen daher, dass aus $Subterms(s) \subseteq S$, $Subterms(t) \subseteq S$, $Subterms(\mathcal{E}) \subseteq S$ und $s \leftrightarrow^*_\mathcal{E} t$ folgt, dass auch $s \equiv t \in CC^S(\mathcal{E})$ ist.

Nach Definition bedeutet $s \leftrightarrow^*_\mathcal{E} t$, dass $s = s_0 \leftrightarrow_\mathcal{E} s_1 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$ für ein $n \geq 0$ gilt. Wir beweisen die Behauptung durch Induktion über n.

Im Induktionsanfang ist $n = 0$. Dann gilt $s = t$ und daher $s \equiv t \in R \cap S \times S \subseteq \mathcal{E}^S_0$.

Im Induktionsschluss ist $n > 0$. Wir betrachten zunächst den Fall, dass in der Herleitung $s = s_0 \leftrightarrow_\mathcal{E} s_1 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$ an mindestens einer Stelle eine Gleichung an der obersten Stelle ϵ angewendet wurde. Es existiert also ein k mit $1 \leq k < n$ und $s_k = u$ und $s_{k+1} = v$ für eine Gleichung $u \equiv v$ oder $v \equiv u$ aus $\mathcal{E}$. Wir haben daher $s = s_0 \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_k = u$ und $v = s_{k+1} \leftrightarrow_\mathcal{E} \ldots \leftrightarrow_\mathcal{E} s_n = t$. Diese beiden Herleitungen sind jeweils kürzer als die ursprüngliche Herleitung und wir haben $Subterms(u) \subseteq S$ und $Subterms(v) \subseteq S$, da u und v Terme aus $\mathcal{E}$ sind. (Dies gilt nur, da $\mathcal{E}$ bloß Grundidentitäten enthält.) Somit folgt aus der Induktionshypothese $s \equiv u \in CC^S(\mathcal{E})$ und $v \equiv t \in CC^S(\mathcal{E})$. Da auch $u \equiv v \in \mathcal{E}^S_1 \subseteq CC^S(\mathcal{E})$ und da $CC^S(\mathcal{E})$ unter Transitivität abgeschlossen ist, folgt $s \equiv t \in CC^S(\mathcal{E})$. Damit haben wir also gezeigt, dass für alle Terme s,t mit $Subterms(s) \subseteq S$, $Subterms(t) \subseteq S$, aus $s \leftrightarrow^*_\mathcal{E} t$ folgt, dass auch $s \equiv t \in CC^S(\mathcal{E})$ gilt, sofern die Herleitung $s \leftrightarrow^*_\mathcal{E} t$ nicht mehr als n Schritte benötigt und an mindestens einer Stelle eine Gleichung an der obersten Stelle ϵ angewendet wurde.

Nun betrachten wir den Fall, dass in der Herleitung $s \leftrightarrow^*_\mathcal{E} t$ keine Gleichung an der obersten Stelle ϵ angewendet wurde. Seien $\pi_1, \ldots, \pi_k$ mit $\pi_i \perp \pi_j$ für $i \neq j$ die *obersten* Stellen in s, an denen in der Herleitung jemals Gleichungen angewendet werden. Somit gilt also $s|_{\pi_i} = p_i$ und $p_i \leftrightarrow^*_\mathcal{E} q_i$ für alle $1 \leq i \leq k$ und $t = s[q_1]_{\pi_1} \ldots [q_k]_{\pi_k}$. Jede Herleitung $p_i \leftrightarrow^*_\mathcal{E} q_i$ ist höchstens so lang wie die Herleitung von $s \leftrightarrow^*_\mathcal{E} t$, d.h., sie benötigt höchstens n Schritte und an mindestens einer Stelle wird eine Gleichung an der obersten Stelle ϵ angewendet. Außerdem gilt $Subterms(p_i) \subseteq Subterms(s) \subseteq S$ und $Subterms(q_i) \subseteq Subterms(t) \subseteq S$. Wie im vorigen Fall zeigt man also $p_i \equiv q_i \in CC^S(\mathcal{E})$. Da $CC^S(\mathcal{E})$ für Terme aus S unter Kongruenz abgeschlossen ist, folgt $s \equiv t \in CC^S(\mathcal{E})$. □

Beispiel 3.2.13 Anhand des Beweises von $\mathsf{f}(\mathsf{m}) \equiv_{\mathcal{E}} \mathsf{g}(\mathsf{k})$ in Bsp. 3.2.6 erkennt man, dass hier nur Gleichungen über den Termen aus $S = \{\mathsf{i}, \mathsf{j}, \mathsf{k}, \mathsf{l}, \mathsf{m}, \mathsf{f}(\mathsf{i}), \mathsf{f}(\mathsf{j}), \mathsf{f}(\mathsf{m}), \mathsf{g}(\mathsf{k}), \mathsf{g}(\mathsf{l})\}$ verwendet wurden. Es gilt also auch $\mathsf{f}(\mathsf{m}) \equiv \mathsf{g}(\mathsf{k}) \ \in \mathcal{E}_9^S \subseteq CC^S(\mathcal{E})$.

Das Verfahren, das gemäß Satz 3.2.12 in endlich vielen Schritten $CC^S(\mathcal{E})$ konstruiert und dann überprüft, ob $s \equiv t \in CC^S(\mathcal{E})$ ist, ist automatisch (mit polynomiellem Aufwand) durchführbar, es terminiert immer und man kann damit entscheiden, ob $s \equiv_{\mathcal{E}} t$ gilt. Insgesamt ergibt sich also das folgende Korollar.

Korollar 3.2.14 (Entscheidbarkeit des Wortproblems bei Grundidentitäten) *Sei $\mathcal{E}$ eine endliche Menge von Grundidentitäten. Dann ist das Wortproblem (d.h., die Frage ob $s \equiv_{\mathcal{E}} t$ für $s, t \in \mathcal{T}(\Sigma)$ gilt) entscheidbar.*

Bei der Berechnung des Kongruenzabschlusses genügt es also, nur Teilterme der Gleichungsmenge und der zu untersuchenden Terme zu betrachten. Dies ist nicht der Fall, wenn man eine Herleitung mittels $\leftrightarrow_{\mathcal{E}}$ finden will. Wie Bsp. 3.2.2 zeigt, muss man hier in den Zwischenschritten auch Terme betrachten, die weder in $\mathcal{E}$ vorkommen noch in den Termen, die man auf Gleichheit untersuchen will. (Der Grund ist, dass hier der Kontext der Terme immer beibehalten wird.) Ansonsten entspricht die Herleitung aus Bsp. 3.2.2 nämlich gerade dem Beweis mit Hilfe des Kongruenzabschlusses aus Bsp. 3.2.6.

Zur praktischen Berechnung des Kongruenzabschlusses ist anstelle von $\mathcal{E}_0^S, \mathcal{E}_1^S, \ldots$ folgende Iteration geeigneter. Hierbei sei für alle Gleichheitssysteme $\mathcal{E}$ die Relation $\Rightarrow_{\mathcal{E}}$ definiert als $s \Rightarrow_{\mathcal{E}} t$ gdw. $s \equiv t \ \in \mathcal{E}$. Wie üblich ist $\Leftrightarrow_{\mathcal{E}}^*$ dann die transitiv-reflexiv-symmetrische Hülle von $\Rightarrow_{\mathcal{E}}$. Sei $Aq(\mathcal{E})$ die Menge aller Gleichungen, die aus $\mathcal{E}$ gemäß (*wiederholter*) Anwendung von Reflexivität, Transitivität und Symmetrie folgen, d.h., $Aq(\mathcal{E}) = \{s \equiv t \mid s \Leftrightarrow_{\mathcal{E}}^* t\}$. Dann sei

- $\mathcal{E}_0^{S'} = Aq(\mathcal{E}) \cap S \times S$
- $\mathcal{E}_{i+1}^{S'} = Aq(\mathcal{E}_i^{S'} \cup C(\mathcal{E}_i^{S'})) \cap S \times S$

Es ist (bei endlichem S) leicht zu zeigen, dass es für alle i ein j gibt, so dass $\mathcal{E}_i^{S'} \subseteq \mathcal{E}_j^S$ und dass es für alle i ein j gibt, so dass $\mathcal{E}_i^S \subseteq \mathcal{E}_j^{S'}$. Somit gilt $CC^S(\mathcal{E}) = \bigcup_{i \in \mathbb{N}} \mathcal{E}_i^{S'}$.

Man kann also anstelle der Iteration $\mathcal{E}_0^S, \mathcal{E}_1^S, \ldots$ auch die Iteration $\mathcal{E}_0^{S'}, \mathcal{E}_1^{S'}, \ldots$ berechnen lassen. Da $\Rightarrow_{\mathcal{E}_i^{S'}}$ jeweils eine Äquivalenzrelation ist, kann man anstelle von $\mathcal{E}_i^{S'}$ auch die Quotientenmenge $S_i = S/_{\Rightarrow_{\mathcal{E}_i^{S'}}}$ berechnen, d.h., die Menge $\{[s]_{\Rightarrow_{\mathcal{E}_i^{S'}}} \mid s \in S\}$. Man hat $S_i = S_{i+1}$ gdw. $\mathcal{E}_i^{S'} = \mathcal{E}_{i+1}^{S'}$. Hierbei ergibt sich dann folgender Algorithmus.

Algorithmus KONGRUENZABSCHLUSS$(\mathcal{E}, s, t)$

Eingabe: Eine endliche Menge von Grundidentitäten $\mathcal{E}$ und zwei Grundterme $s, t \in \mathcal{T}(\Sigma)$.

Ausgabe: "*True*", falls $s \equiv_{\mathcal{E}} t$ und sonst "*False*".

1. Sei $S = Subterms(\mathcal{E}) \cup Subterms(s) \cup Subterms(t)$.
2. Sei $L = \{\{s,t\} \mid s \equiv t \in \mathcal{E}\} \cup \{\{s\} \mid s \in S\}$.
3. Vereinige alle Mengen $M_1, M_2 \in L$ mit $M_1 \cap M_2 \neq \varnothing$.
4. Sei $K = L \cup \{\{f(s_1,\ldots,s_n), f(t_1,\ldots,t_n)\} \mid f \in \Sigma$, es ex. $M_i \in L$ mit $s_i, t_i \in M_i$, $f(s_1,\ldots,s_n), f(t_1,\ldots,t_n) \in S\}$.
5. Vereinige alle Mengen $M_1, M_2 \in K$ mit $M_1 \cap M_2 \neq \varnothing$.
6. Falls $K \neq L$ dann setze $L = K$ und gehe zu 4.
7. Falls es ein $M \in L$ gibt mit $s \in M$ und $t \in M$, dann gib "*True*" aus. Sonst gib "*False*" aus.

Der obige Algorithmus lässt sich natürlich noch etwas verbessern, indem man bereits vor der "Kongruenz–Bildung" in Schritt 4 jeweils überprüft, ob bereits s und t zusammen in einer Menge $M \in L$ auftreten. In diesem Fall kann man sofort "*True*" ausgeben und den Algorithmus beenden.

Satz 3.2.15 (Korrektheit des Algorithmus)
Der Algorithmus KONGRUENZABSCHLUSS *terminiert immer und ist korrekt.*

Beweisskizze. Die Terminierung folgt analog zu Lemma 3.2.11, denn da S endlich ist und L in jedem Schleifendurchlauf wächst, gilt irgendwann $K = L$. Die Korrektheit gilt aus folgendem Grund: Nach Schritt 3 ist $L = S_0$, d.h., es gilt $s,t \in M$ für ein $M \in L$ gdw. $s \Leftrightarrow^*_{\mathcal{E}} t$ und $s,t \in S$. Mit anderen Worten, man hat $\mathcal{E}_0^{S'} = \{s \equiv t \mid$ es existiert ein $M \in L$ mit $s,t \in M\}$. (Die Vereinigung aller nicht-disjunkten Mengen erzeugt gerade den transitiv-reflexiv-symmetrischen Abschluss.) Nach Schritt 5 gilt im i-ten Schleifendurchlauf $K = S_i$, d.h., es gilt $s,t \in M$ für ein $M \in K$ gdw. $s \Leftrightarrow^*_{\mathcal{E}_{i-1}^{S'} \cup C(\mathcal{E}_{i-1}^{S'})} t$ und $s,t \in S$. Man hat somit $\mathcal{E}_i^{S'} = \{s \equiv t \mid$ es existiert ein $M \in K$ mit $s,t \in M\}$ (dies zeigt man durch Induktion über i). In Schritt 7 gilt somit $L = S/_{\Rightarrow_{CC^S(\mathcal{E})}}$, d.h., es existiert ein $M \in L$ mit $s,t \in M$ gdw. $s \equiv t \in CC^S(\mathcal{E})$. Aufgrund der Korrektheit und Vollständigkeit des Kongruenzabschlusses bzgl. S (Satz 3.2.12) ist dies gleichbedeutend zu $s \equiv_{\mathcal{E}} t$. □

Beispiel 3.2.16 Der obige Algorithmus arbeitet wie folgt in unserem Beispiel. Hierbei ist wieder $\mathcal{E} = \{\mathsf{i} \equiv \mathsf{j}, \mathsf{k} \equiv \mathsf{l}, \mathsf{f(i)} \equiv \mathsf{g(k)}, \mathsf{j} \equiv \mathsf{f(j)}, \mathsf{m} \equiv \mathsf{g(l)}\}$, $s = \mathsf{f(m)}$ und $t = \mathsf{g(k)}$.

- Im ersten Schritt erzeugt man $S = \{\mathsf{i}, \mathsf{j}, \mathsf{k}, \mathsf{l}, \mathsf{m}, \mathsf{f(i)}, \mathsf{f(j)}, \mathsf{f(m)}, \mathsf{g(k)}, \mathsf{g(l)}\}$.
- Im zweiten Schritt setzt man L auf die Menge, die $\{\mathsf{i,j}\}, \{\mathsf{k,l}\}, \{\mathsf{f(i)}, \mathsf{g(k)}\}, \{\mathsf{j}, \mathsf{f(j)}\}$, $\{\mathsf{m}, \mathsf{g(l)}\}, \{\mathsf{f(m)}\}$ sowie $\{q\}$ für alle $q \in S$ enthält.
- Im dritten Schritt vereinigt man alle nicht-disjunkten Mengen von L wie z.B. $\{\mathsf{i,j}\}$ und $\{\mathsf{j}, \mathsf{f(j)}\}$. Es ergibt sich $L = \{\{\mathsf{i}, \mathsf{j}, \mathsf{f(j)}\}, \{\mathsf{k,l}\}, \{\mathsf{f(i)}, \mathsf{g(k)}\}, \{\mathsf{m}, \mathsf{g(l)}\}, \{\mathsf{f(m)}\}\}$. Dies ist die Quotientenmenge zur Relation $\Leftrightarrow^*_{\mathcal{E}}$, bei der zwei Terme gleich sind, wenn dies aufgrund der (wiederholten) Reflexivität, Transitivität und Symmetrie aus $\mathcal{E}$ folgt.
- Nun setzt man K auf L und erweitert es um Mengen $\{\mathsf{f}(t_1), \mathsf{f}(t_2)\}$ und $\{\mathsf{g}(t_1), \mathsf{g}(t_2)\}$, wobei t_1 und t_2 jeweils aus denselben Mengen von L stammen müssen. Es werden aber nur solche Terme betrachtet, die in S enthalten sind. Neben einelementigen Mengen (bei denen also $t_1 = t_2$ ist) enthält K dann die zusätzlichen Mengen $\{\mathsf{f(i)}, \mathsf{f(j)}\}$ und $\{\mathsf{g(k)}, \mathsf{g(l)}\}$.

- In Schritt 5 vereinigt man alle nicht-disjunkten Mengen in K und erhält somit $K = \{\{\mathsf{i}, \mathsf{j}, \mathsf{f(j)}, \mathsf{f(i)}, \mathsf{g(k)}, \mathsf{g(l)}, \mathsf{m}\}, \{\mathsf{k}, \mathsf{l}\}, \{\mathsf{f(m)}\}\}$. Da $K \neq L$ gilt, wird in Schritt 6 L auf diese Menge gesetzt und man geht zurück zu Schritt 4.

- Man setzt nun K wieder auf L und erweitert es (neben einelementigen Mengen) um $\{\mathsf{f(i)}, \mathsf{f(j)}\}, \{\mathsf{f(i)}, \mathsf{f(m)}\}, \{\mathsf{f(j)}, \mathsf{f(m)}\}, \{\mathsf{g(k)}, \mathsf{g(l)}\}$. In Schritt 5 vereinigt man alle nicht-disjunkten Mengen in K und erhält somit $K = \{\{\mathsf{i}, \mathsf{j}, \mathsf{f(j)}, \mathsf{f(i)}, \mathsf{g(k)}, \mathsf{g(l)}, \mathsf{m}, \mathsf{f(m)}\}, \{\mathsf{k}, \mathsf{l}\}\}$. Da $K \neq L$ gilt, wird in Schritt 6 L auf diese Menge gesetzt und man geht zurück zu Schritt 4.

- Das nochmalige Durchlaufen von Schritt 4 und 5 führt dazu, dass K und L identisch sind. Man erreicht somit Schritt 7. Da es eine Menge $\{\mathsf{i}, \mathsf{j}, \mathsf{f(j)}, \mathsf{f(i)}, \mathsf{g(k)}, \mathsf{g(l)}, \mathsf{m}, \mathsf{f(m)}\}$ in K gibt, die sowohl $\mathsf{f(m)}$ als auch $\mathsf{g(k)}$ enthält, gibt der Algorithmus "*True*" aus.

Das in diesem Abschnitt vorgestellte Verfahren ist allerdings tatsächlich nur für Grundidentitäten verwendbar. Bei $\mathcal{E} = \{\mathsf{f(f}(x)) \equiv \mathsf{g}(x)\}$ gilt z.B. $\mathsf{f(g(a))} \equiv_{\mathcal{E}} \mathsf{g(f(a))}$, aber dies lässt sich nur mit Hilfe der Gleichungen $\mathsf{f(g(a))} \equiv_{\mathcal{E}} \mathsf{f(f(f(a)))}$ und $\mathsf{f(f(f(a)))} \equiv \mathsf{g(f(a))}$ und der Transitivität zeigen. Der Term $\mathsf{f(f(f(a)))}$ ist aber nicht in $Subterms(\mathcal{E}) \cup Subterms(\mathsf{f(g(a))}) \cup Subterms(\mathsf{g(f(a))})$ enthalten.

3.3 Termersetzungssysteme

Im vorigen Abschnitt haben wir gesehen, wie man das Wortproblem für Grundidentitäten entscheiden kann, d.h. für Gleichungssysteme $\mathcal{E}$, die keine Variablen enthalten. In vielen Anwendungen treten aber sowohl in den Axiomen als auch in den zu untersuchenden Gleichungen Variablen auf. Man geht hierbei davon aus, dass die Axiome *für alle* Belegungen der darin vorkommenden Variablen gelten und man will wissen, ob die zu untersuchende Gleichung *für alle* Belegungen der darin vorkommenden Variablen aus den Axiomen folgt.

Wie in Abschnitt 3.1 gezeigt wurde, kann man dieses Wortproblem mit Hilfe der Beweisrelation $\leftrightarrow^*_{\mathcal{E}}$ untersuchen. Das Problem dabei ist aber, dass man nicht weiß, welche Gleichung an welcher Stelle angewendet werden soll, so dass sich ein in der Praxis nicht tolerierbarer und ggf. unendlicher Suchaufwand ergibt.

Will man den Aufwand des Suchbaumverfahrens aus Abschnitt 3.1 senken, so müssen die Indeterminismen der Beweisrelation so verringert werden, dass man schließlich den Verzweigungsgrad 1 erhält. Der erste Schritt besteht darin, dass die *Richtung* der Gleichungen des Gleichungssystems festgelegt wird. Damit kann man bei Teiltermersetzungen die Gleichungen nur noch in eine Richtung anwenden. Wir schreiben daher nun einen Pfeil "$\rightarrow$" anstelle des Gleichheitszeichens "$\equiv$" und bezeichnen die gerichteten Gleichungen als *Regeln*. Dies führt zu dem grundlegenden Begriff des *Termersetzungssystems*.

Definition 3.3.1 (Termersetzungssystem) *Für $l, r \in \mathcal{T}(\Sigma, \mathcal{V})$ heißt ein Ausdruck*

$$l \rightarrow r$$

(Termersetzungs-)regel *(über Σ und $\mathcal{V}$), sofern*

- $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ *und*

- $l \notin \mathcal{V}$.

Eine Menge $\mathcal{R}$ von Regeln wird Termersetzungssystem *(TES, engl. term rewriting system) über Σ und $\mathcal{V}$ genannt.*

Für ein Termersetzungssystem $\mathcal{R}$ ist die (Term-)ersetzungsrelation $\to_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$ *definiert durch*

$$s \to_{\mathcal{R}} t \quad \textit{gdw.} \quad s|_\pi = l\sigma \;\; \textit{und} \;\; t = s[r\sigma]_\pi$$

für eine Stelle $\pi \in Occ(s)$, eine Regel $l \to r \in \mathcal{R}$ und ein $\sigma \in SUB(\Sigma, \mathcal{V})$. Man nennt $s \to_{\mathcal{R}} t$ einen (Term-)ersetzungsschritt*, bei dem s an der Stelle π* reduziert *wird. Ein Teilterm $s|_\pi$, auf den die linke Seite einer Regel matcht, heißt* Redex *(für "reducible expression"). Wenn $\mathcal{R}$ aus dem Zusammenhang ersichtlich ist, schreiben wir manchmal auch "$\to$" anstelle von "$\to_{\mathcal{R}}$". Anstelle von "$s \to_{\mathcal{R}} t$" schreiben wir außerdem gelegentlich auch "$t \leftarrow_{\mathcal{R}} s$".*

Wie bei den Gleichungssystemen werden wir uns auch bei TESen im Folgenden auf endliche Mengen beschränken. Die Termersetzungsrelation $\to_{\mathcal{R}}$ ist genauso definiert wie die Ersetzungsrelation $\to_{\mathcal{E}}$ für eine Menge von Gleichungen. Der einzige Unterschied ist, dass wir jetzt mehr an der transitiv-reflexiven Hülle $\to^*$ interessiert sind als an der transitiv-reflexiv-symmetrischen Hülle $\leftrightarrow^*$, d.h., wir wenden die Regeln eines Termersetzungssystems nur in eine Richtung an, wohingegen Gleichungen typischerweise in beide Richtungen angewendet werden können.

Durch das Richten der Gleichungen wird der Indeterminismus des Suchverfahrens aus Abschnitt 3.1 bereits reduziert. Die Bedingung $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ entspricht dann der "Variablenbedingung", die dort diskutiert wurde. Diese Variablenbedingung verlangte, dass auf beiden Seiten einer Gleichung genau dieselben Variablen vorkommen. Dies hat den Vorteil, dass bei Anwendung einer Gleichung der Matcher (und damit die Instantiierung der anderen Seite der Gleichung) bereits eindeutig festgelegt ist. Da Gleichungen nun aber nur noch von links nach rechts angewendet werden können, muss man bloß noch verlangen, dass alle Variablen auf der rechten Seite auch auf der linken Seite auftreten. Damit ist bei einer Anwendung einer Termersetzungsregel die Instantiierung der rechten Seite bereits eindeutig festgelegt.

Auch die Bedingung $l \notin \mathcal{V}$ dient dazu, den Indeterminismus zu reduzieren. Regeln mit einer Variablen als linke Seite wären immer (d.h. auf jeden Term) anwendbar. Durch die Bedingung, dass eine linke Seite zumindest mit einem Funktionssymbol beginnen muss, wird ihre Anwendbarkeit etwas eingeschränkt.

Beispiel 3.3.2 Sowohl die Additionsgleichungen als auch die Gruppenaxiome erfüllen die Variablenbedingungen. Sie lassen sich daher nun beide als TES schreiben. (Selbstverständlich stellt sich hierbei im allgemeinen die Frage, in welche Richtung solche Gleichungen zu orientieren sind, wenn dies nicht bereits durch die Variablenbedingungen erzwungen wird.)

$$\mathsf{plus}(\mathcal{O}, y) \to y \qquad (3.1)$$

$$\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathsf{plus}(x, y)) \qquad (3.2)$$

$$\mathsf{f}(x, \mathsf{f}(y, z)) \to \mathsf{f}(\mathsf{f}(x, y), z) \qquad (3.3)$$

$$\mathsf{f}(x, \mathsf{e}) \to x \qquad (3.4)$$

$$\mathsf{f}(x, \mathsf{i}(x)) \to \mathsf{e} \qquad (3.5)$$

Wenn $\mathcal{R} = \{(3.1), (3.2)\}$ ist, so erhält man z.B. die folgende Folge von Ersetzungsschritten. Die Redexe sind hierbei jeweils unterstrichen. (Im allgemeinen kann ein Term natürlich

mehrere Redexe besitzen, die nicht unbedingt alle reduziert werden.)

$$\begin{aligned}\underline{\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})),\mathsf{succ}(\mathcal{O}))} &\to_{\mathcal{R}} \mathsf{succ}(\underline{\mathsf{plus}(\mathsf{succ}(\mathcal{O}),\mathsf{succ}(\mathcal{O}))})\\ &\to_{\mathcal{R}} \mathsf{succ}(\mathsf{succ}(\underline{\mathsf{plus}(\mathcal{O},\mathsf{succ}(\mathcal{O}))}))\\ &\to_{\mathcal{R}} \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})))\end{aligned}$$

Man erhält also die Reduktion $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})),\mathsf{succ}(\mathcal{O})) \to^*_{\mathcal{R}} \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})))$.

Die Sprache der Termersetzungssysteme ist also fast identisch mit der Sprache der Gleichungssysteme, aber Termersetzungssysteme stellen eine Programmiersprache mit einer bestimmten Auswertungsrelation (der Termersetzungsrelation) dar. Man kann zeigen, dass diese Programmiersprache Turing-vollständig ist, d.h., jedes berechenbare Programm lässt sich mit Hilfe von TESen schreiben. In der Tat beruhen insbesondere funktionale Programmiersprachen direkt auf der Sprache der TESe (wobei natürlich unterschiedliche und ergänzende syntaktische Konstrukte verwendet werden).

Manchmal betrachtet man spezielle Termersetzungssysteme, bei denen die Regeln lediglich einstellige Funktionssymbole enthalten. Ein Beispiel ist das TES mit der Regel

$$\mathsf{f}(\mathsf{f}(x)) \;\to\; \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$$

Die Regeln solcher TESe kann man kürzer schreiben, indem man die Klammern und die innerste Variable weglässt; man erhält dann

$$\mathsf{ff} \;\to\; \mathsf{fgf}.$$

Anstelle der Auswertung $\mathsf{f}(\mathsf{f}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x)))) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x)))))$ schreibt man dann kürzer

$$\mathsf{fff} \to \mathsf{ffgf} \to \mathsf{fgfgf}.$$

Solche Regelsysteme bezeichnet man auch als *Wortersetzungssysteme* (engl. string rewriting system) oder *Semi-Thue Systeme*. Hier kann man die Terme als Worte auffassen und die Regeln entsprechen Grammatikregeln, die sagen, wie man bestimmte Teilworte ersetzen kann. Wir werden den Spezialfall der Wortersetzungssysteme nicht weiter gesondert betrachten; hierzu sei auf [BA95, Kap. 3] und [Ave95, Kap. 2] verwiesen.

Aus Lemma 3.1.13 folgt sofort, dass die Termersetzungsrelation $\to_{\mathcal{R}}$ ebenso wie die Relation $\to^*_{\mathcal{R}}$ stabil und monoton ist. Darüber hinaus ist $\to^*_{\mathcal{R}}$ natürlich reflexiv und transitiv, aber nicht symmetrisch.

Anstelle von Gleichungssystemen $\mathcal{E}$ wollen wir im Folgenden mit TESen $\mathcal{R}$ arbeiten. Hierzu muss das TES $\mathcal{R}$ aber natürlich dem Gleichungssystem $\mathcal{E}$ "entsprechen". Hierzu definiert man den Begriff der "Äquivalenz".

Definition 3.3.3 (Äquivalenz von Gleichungssystem und TES) *Sei $\mathcal{E}$ ein Gleichungssystem und $\mathcal{R}$ ein TES über Σ und $\mathcal{V}$. Dann ist $\mathcal{R}$* äquivalent *zu $\mathcal{E}$ gdw. $\leftrightarrow^*_{\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}}$.*

Wir sind ja daran interessiert, das Wortproblem $s \equiv_{\mathcal{E}} t$ für Gleichungssysteme $\mathcal{E}$ zu lösen. Aufgrund des Satzes von Birkhoff (Satz 3.1.14) bedeutet Äquivalenz zwischen $\mathcal{R}$ und $\mathcal{E}$, dass $\leftrightarrow^*_{\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}} = \equiv_{\mathcal{E}}$ gilt. Sofern ein TES $\mathcal{R}$ also äquivalent zu $\mathcal{E}$ ist, lässt sich das

Wortproblem für $\mathcal{E}$ dadurch lösen, dass man eine Folge von $\mathcal{R}$-Ersetzungsschritten sucht, so dass sich s und t ineinander überführen lassen. Allerdings können bei dieser Folge die Regeln von $\mathcal{R}$ noch in beide Richtungen angewendet werden.

Zunächst wollen wir aber die Frage untersuchen, wie man zu einem Gleichungssystem $\mathcal{E}$ ein äquivalentes TES $\mathcal{R}$ findet. Falls sich $\mathcal{E}$ durch Ersetzung der Gleichheitszeichen durch Pfeile direkt in ein TES überführen lässt, so ist dieses TES trivialerweise äquivalent zu $\mathcal{E}$. Satz 3.3.4 zeigt einen allgemeineren Zusammenhang zwischen Gleichungssystemen und dazu äquivalenten TESen. Er macht deutlich, dass man nur die Regeln in $\mathcal{R}$ und die Gleichungen in $\mathcal{E}$ untersuchen muss (und nicht alle Paare von Termen, die sich in der transitiv-reflexiv-symmetrischen Hülle der beiden Relationen befinden), um festzustellen, ob das TES $\mathcal{R}$ äquivalent zu $\mathcal{E}$ ist.

Satz 3.3.4 (Zusammenhang zwischen Gleichungssystem und TES) *Sei $\mathcal{E}$ ein Gleichungssystem und $\mathcal{R}$ ein TES über Σ und $\mathcal{V}$. $\mathcal{R}$ ist äquivalent zu $\mathcal{E}$ gdw.*

- *$l \leftrightarrow^*_{\mathcal{E}} r$ für alle Regeln $l \to r \in \mathcal{R}$ ("$\mathcal{R}$ ist* korrekt *für $\mathcal{E}$") und*
- *$s \leftrightarrow^*_{\mathcal{R}} t$ für alle Gleichungen $s \equiv t \in \mathcal{E}$ ("$\mathcal{R}$ ist* adäquat *für $\mathcal{E}$").*

Beweis. Aus der Äquivalenz von $\mathcal{R}$ und $\mathcal{E}$ folgt die Korrektheit sofort. Da außerdem für alle Gleichungen $s \equiv t \in \mathcal{E}$ auch $s \leftrightarrow^*_{\mathcal{E}} t$ gilt, folgt aus der Äquivalenz $s \leftrightarrow^*_{\mathcal{R}} t$ (d.h., $\mathcal{R}$ ist adäquat für $\mathcal{E}$).

Nun zeigen wir, dass aus der Korrektheit und Adäquatheit die Äquivalenz folgt. Wenn für alle Regeln $l \to r \in \mathcal{R}$ jeweils $l \leftrightarrow^*_{\mathcal{E}} r$ gilt, so folgt auch aus $s \leftrightarrow^*_{\mathcal{R}} t$ ebenfalls $s \leftrightarrow^*_{\mathcal{E}} t$ aufgrund der Abgeschlossenheit von $\leftrightarrow^*_{\mathcal{E}}$ unter Substitutionen und Kontexten (formal ist hierbei eine Induktion über die Anzahl der Ersetzungsschritte bei $s \leftrightarrow^*_{\mathcal{R}} t$ nötig). Aus der Korrektheit folgt somit $\leftrightarrow^*_{\mathcal{R}} \subseteq \leftrightarrow^*_{\mathcal{E}}$.

Umgekehrt folgt aus der Adäquatheit, dass $s \leftrightarrow^*_{\mathcal{E}} t$ jeweils auch $s \leftrightarrow^*_{\mathcal{R}} t$ impliziert, denn $s \leftrightarrow^*_{\mathcal{R}} t$ ist ebenfalls abgeschlossen unter Substitutionen und Kontexten (formal ist hier eine Induktion über die Anzahl der Ersetzungsschritte bei $s \leftrightarrow^*_{\mathcal{E}} t$ nötig). Somit erhält man $\leftrightarrow^*_{\mathcal{E}} \subseteq \leftrightarrow^*_{\mathcal{R}}$. □

Beispiel 3.3.5 Das TES $\{(3.1), (3.2)\}$ ist trivialerweise äquivalent zu der Gleichungsmenge $\{\mathsf{plus}(\mathcal{O}, y) \equiv y, \mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathsf{plus}(x, y))\}$.

Als weiteres Beispiel betrachten wir das Gleichheitssystem $\mathcal{E} = \{\mathsf{b} \equiv \mathsf{c}, \mathsf{b} \equiv \mathsf{a}, \mathsf{f}(\mathsf{a}) \equiv \mathsf{f}(\mathsf{f}(\mathsf{a}))\}$ und $\mathcal{R} = \{\mathsf{c} \to \mathsf{b}, \mathsf{a} \to \mathsf{b}, \mathsf{f}(\mathsf{f}(\mathsf{b})) \to \mathsf{f}(\mathsf{b})\}$. Dann ist $\mathcal{R}$ für $\mathcal{E}$ korrekt, denn

- $\mathsf{c} \leftarrow_{\mathcal{E}} \mathsf{b}$
- $\mathsf{a} \leftarrow_{\mathcal{E}} \mathsf{b}$
- $\mathsf{f}(\mathsf{f}(\mathsf{b})) \to_{\mathcal{E}} \mathsf{f}(\mathsf{f}(\mathsf{a})) \leftarrow_{\mathcal{E}} \mathsf{f}(\mathsf{a}) \leftarrow_{\mathcal{E}} \mathsf{f}(\mathsf{b})$

$\mathcal{R}$ ist auch adäquat für $\mathcal{E}$, denn

- $\mathsf{b} \leftarrow_{\mathcal{R}} \mathsf{c}$,
- $\mathsf{b} \leftarrow_{\mathcal{R}} \mathsf{a}$,

- $\mathsf{f(a)} \to_{\mathcal{R}} \mathsf{f(b)} \leftarrow_{\mathcal{R}} \mathsf{f(f(b))} \leftarrow_{\mathcal{R}} \mathsf{f(f(a))}$

Nach Satz 3.3.4 ist $\mathcal{R}$ also äquivalent zu $\mathcal{E}$. Hingegen wäre z.B. das TES $\{\mathsf{c} \to \mathsf{a}, \mathsf{f(f(b))} \to \mathsf{f(b)}\}$ zwar korrekt, aber nicht adäquat und somit auch nicht äquivalent zu $\mathcal{E}$.

Bislang ist noch unklar, warum man nicht direkt eine gerichtete Version der Gleichheiten $\mathcal{E}$ als äquivalentes TES $\mathcal{R}$ verwendet. Der Grund liegt daran, dass allein durch Richten von Gleichungen der Beweisaufwand immer noch viel zu groß ist. Man muss momentan Gleichungen ja immer noch in beide Richtungen anwenden, um die Äquivalenz zu $\equiv_{\mathcal{E}}$ zu erreichen. Außerdem gibt es viele verschiedene Möglichkeiten, Terme mit einem TES zu reduzieren, so dass immer noch viel zu viele Indeterminismen bleiben.

Die Idee ist stattdessen, zur Überprüfung von $s \equiv_{\mathcal{E}} t$ bzw. von $s \leftrightarrow^*_{\mathcal{R}} t$ (für ein zu $\mathcal{E}$ äquivalentes TES $\mathcal{R}$) nicht mehr Regeln in beliebiger Richtung anzuwenden, sondern wir reduzieren zuerst s und t soweit wie möglich. Damit erhält man Auswertungsfolgen der Art

$$s \to_{\mathcal{R}} s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \ldots s_n \quad \text{und} \quad t_m \leftarrow_{\mathcal{R}} \ldots t_2 \leftarrow_{\mathcal{R}} t_1 \leftarrow_{\mathcal{R}} t,$$

wobei s_n und t_m nicht weiter mit $\to_{\mathcal{R}}$ reduzierbar sind. Anschließend überprüfen wir, ob die beiden Terme s_n und t_m, die wir erhalten haben, identisch (d.h. syntaktisch gleich) sind.

Beispiel 3.3.6 Betrachten wir wieder das TES $\mathcal{R}$ mit den Regeln (3.1) und (3.2). Unsere Frage war, ob aus den plus-Axiomen $\mathcal{E}$ die Gleichung $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ folgt. Um dieses Wortproblem zu lösen, gehen wir wie folgt vor: Wir reduzieren zuerst beide Terme in der Gleichung soweit wie möglich und überprüfen dann, ob die erhaltenen Terme identisch sind. Im Beispiel ergibt sich

$$\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \to_{\mathcal{R}} \mathsf{succ}(\mathsf{plus}(\mathsf{succ}(\mathcal{O}), x)) \to_{\mathcal{R}} \mathsf{succ}(\mathsf{succ}(\mathsf{plus}(\mathcal{O}, x))) \to_{\mathcal{R}} \mathsf{succ}(\mathsf{succ}(x))$$

und

$$\mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x)) \to_{\mathcal{R}} \mathsf{succ}(\mathsf{plus}(\mathcal{O}, \mathsf{succ}(x))) \to_{\mathcal{R}} \mathsf{succ}(\mathsf{succ}(x)).$$

Somit erhält man also in der Tat $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \leftrightarrow^*_{\mathcal{R}} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ bzw. $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv_{\mathcal{E}} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ aufgrund der Äquivalenz zu $\mathcal{E}$.

Das oben skizzierte Verfahren ist mit Sicherheit korrekt, denn aus der Existenz von zwei Auswertungsfolgen zum selben Ergebnis folgt immer $s \leftrightarrow^*_{\mathcal{R}} t$ und somit $s \equiv_{\mathcal{E}} t$. Es stellen sich aber bislang noch zwei Probleme:

1. Wenn wir s und t solange wie möglich reduzieren wollen, so könnte es sein, dass diese Auswertung nicht *terminiert*. Hierbei gibt es zum einen die Möglichkeit, dass *keine* der möglichen Auswertungen von s bzw. t terminiert. Es kann aber auch sein, dass es zwar terminierende, aber auch nicht-terminierende Auswertungen dieser Terme gibt. Da man die Folge der Auswertungsschritte frei wählen kann, besteht dann immer noch die Gefahr, dass man dennoch die unendliche Ersetzungsfolge wählt. Um dies zu verhindern, muss man sich auf sogenannte *terminierende Termersetzungssysteme* beschränken.

2. Es könnte sein, dass zwar $s \leftrightarrow^*_{\mathcal{R}} t$ gilt, dass man dazu aber die Richtung der Reduktion *mehrmals* bzw. *auf andere Weise* wechseln muss. Mit anderen Worten, es existiert möglicherweise trotzdem kein Term q mit $s \rightarrow^*_{\mathcal{R}} q$ und $q \leftarrow^*_{\mathcal{R}} t$. Darüber hinaus könnte es sein, dass s und t auf verschiedene Arten reduziert werden könnten, so dass es an der Wahl der "richtigen" Reduktionsfolge liegt, ob man den gewünschten Beweis findet oder nicht. Um diese Probleme zu verhindern, muss man sich auf sogenannte *konfluente Termersetzungssysteme* einschränken.

Im Folgenden wollen wir diese Probleme näher skizzieren und die gewünschten Eigenschaften formal definieren.

Beispiel 3.3.7 Um das Problem der Nicht-Terminierung zu illustrieren, betrachten wir wieder die Gleichungsmenge $\mathcal{E} = \{\mathsf{b} \equiv \mathsf{c}, \mathsf{b} \equiv \mathsf{a}, \mathsf{f}(\mathsf{a}) \equiv \mathsf{f}(\mathsf{f}(\mathsf{a}))\}$ aus Bsp. 3.3.5 und das äquivalente TES $\mathcal{R}_1 = \{\mathsf{b} \rightarrow \mathsf{c}, \mathsf{b} \rightarrow \mathsf{a}, \mathsf{f}(\mathsf{a}) \rightarrow \mathsf{f}(\mathsf{f}(\mathsf{a}))\}$, das durch Ersetzung des Gleichheitszeichens durch "$\rightarrow$" entsteht.

Um das Wortproblem $\mathsf{f}(\mathsf{a}) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{c})$ zu untersuchen, würden wir nun $\mathsf{f}(\mathsf{a})$ und $\mathsf{f}(\mathsf{c})$ jeweils soweit wie möglich reduzieren. Der Term $\mathsf{f}(\mathsf{c})$ kann nicht weiter reduziert werden, aber die (einzige) Auswertungsfolge von $\mathsf{f}(\mathsf{a})$ ist unendlich. Es ergibt sich

$$\mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}_1} \mathsf{f}^2(\mathsf{a}) \rightarrow_{\mathcal{R}_1} \mathsf{f}^3(\mathsf{a}) \rightarrow_{\mathcal{R}_1} \ldots$$

Somit findet man niemals einen nicht mehr reduzierbaren Term und das Verfahren terminiert nicht.

Alternativ könnte man auch das ebenfalls äquivalente TES $\mathcal{R}_2 = \{\mathsf{b} \rightarrow \mathsf{c}, \mathsf{b} \rightarrow \mathsf{a}, \mathsf{f}(\mathsf{a}) \rightarrow \mathsf{f}(\mathsf{f}(\mathsf{b}))\}$ betrachten. Nun gibt es Reduktionen von $\mathsf{f}(\mathsf{a})$ zu den Termen $\mathsf{f}^2(\mathsf{c})$, $\mathsf{f}^3(\mathsf{c})$, etc., die nicht mehr weiter reduzierbar sind. Allerdings kann die Folge von Ersetzungsschritten beliebig gewählt werden und somit ist nicht sicher gestellt, ob solch eine endliche Folge gewählt wird oder ob man bei der Reduktion von $\mathsf{f}(\mathsf{a})$ die unendliche Folge $\mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}_2} \mathsf{f}^2(\mathsf{b}) \rightarrow_{\mathcal{R}_2} \mathsf{f}^2(\mathsf{a}) \rightarrow_{\mathcal{R}_2} \mathsf{f}^3(\mathsf{b}) \rightarrow_{\mathcal{R}_2} \ldots$ wählt.

Um die anderen oben angesprochenen Probleme zu illustrieren, verwenden wir nun ein anderes ebenfalls äquivalentes TES $\mathcal{R}_3 = \{\mathsf{b} \rightarrow \mathsf{c}, \mathsf{b} \rightarrow \mathsf{a}, \mathsf{f}(\mathsf{f}(\mathsf{a})) \rightarrow \mathsf{f}(\mathsf{a})\}$, bei dem alle Auswertungsfolgen endlich sind. Wenn wir wieder das Wortproblem $\mathsf{f}(\mathsf{a}) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{c})$ untersuchen wollen, stellt sich die Schwierigkeit, dass zwar $\mathsf{f}(\mathsf{a}) \leftrightarrow^*_{\mathcal{R}_3} \mathsf{f}(\mathsf{c})$ gilt (denn $\mathsf{f}(\mathsf{a}) \leftarrow_{\mathcal{R}_3} \mathsf{f}(\mathsf{b}) \rightarrow_{\mathcal{R}_3} \mathsf{f}(\mathsf{c})$), aber diese Herleitung lässt sich nicht dadurch finden, dass man erst die beiden Terme $\mathsf{f}(\mathsf{a})$ und $\mathsf{f}(\mathsf{c})$ reduziert und dann überprüft, ob die entstandenen Terme identisch sind. Sowohl $\mathsf{f}(\mathsf{a})$ als auch $\mathsf{f}(\mathsf{c})$ sind nämlich nicht weiter reduzierbar und dennoch nicht identisch.

Schließlich gibt es auch den Fall, dass man das Wortproblem zwar auf die gewünschte Art lösen kann, es aber mehrere Ersetzungsmöglichkeiten gibt, und der Erfolg des Beweises davon abhängt, ob man die "richtige" Ersetzung wählt. Hierzu betrachten wir wieder das TES $\mathcal{R}_3$ und das Wortproblem $\mathsf{f}^2(\mathsf{b}) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{a})$. Hierbei kann man $\mathsf{f}^2(\mathsf{b})$ bei der richtigen Wahl der Ersetzungsschritte zu $\mathsf{f}(\mathsf{a})$ reduzieren, so dass die Gleichheit nachgewiesen werden kann. Man kann aber $\mathsf{f}^2(\mathsf{b})$ auch zu $\mathsf{f}^2(\mathsf{c})$ reduzieren und dann scheitert der Nachweis. Da man die Reduktion frei wählen kann, ist nicht gesagt, welche der möglichen Ersetzungsschritte man wählt. Auch das TES $\mathcal{R}_4 = \{\mathsf{c} \rightarrow \mathsf{b}, \mathsf{a} \rightarrow \mathsf{b}, \mathsf{f}(\mathsf{f}(\mathsf{a})) \rightarrow \mathsf{f}(\mathsf{a})\}$ ist ungeeignet, denn obwohl $\mathsf{f}^2(\mathsf{b}) \leftrightarrow^*_{\mathcal{R}_4} \mathsf{f}(\mathsf{b})$ gilt, kann man dies nicht auf die gewünschte Art beweisen, da sowohl $\mathsf{f}^2(\mathsf{b})$ als auch $\mathsf{f}(\mathsf{b})$ irreduzibel sind.

Die Lösung für diese Probleme ist, nicht ein TES wie $\mathcal{R}_1$ - $\mathcal{R}_4$ zu verwenden, sondern z.B. das TES $\mathcal{R} = \{\mathsf{c} \to \mathsf{b}, \mathsf{a} \to \mathsf{b}, \mathsf{f}(\mathsf{f}(\mathsf{b})) \to \mathsf{f}(\mathsf{b})\}$ aus Bsp. 3.3.5. Dieses TES hat die Eigenschaft, dass jede Folge von $\mathcal{R}$-Ersetzungsschritten *terminiert* und dass das Ergebnis der Reduktion *eindeutig* ist (d.h., es ist *konfluent*). Hier muss man $\mathsf{f}(\mathsf{a})$ und $\mathsf{f}(\mathsf{c})$ beide zu $\mathsf{f}(\mathsf{b})$ reduzieren und kann somit die Gleichheit von $\mathsf{f}(\mathsf{a})$ und $\mathsf{f}(\mathsf{c})$ nachweisen. Ebenso kann man $\mathsf{f}^2(\mathsf{b}) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{a})$ nachweisen, da man beide Terme nur zu $\mathsf{f}(\mathsf{b})$ reduzieren kann, und $\mathsf{f}^2(\mathsf{b}) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{b})$ folgt, da $\mathsf{f}^2(\mathsf{b})$ zu $\mathsf{f}(\mathsf{b})$ reduzierbar ist.

Man erkennt also, dass es nicht immer sinnvoll ist, ein äquivalentes TES zu verwenden, das direkt durch Richten der ursprünglichen Gleichungen entsteht. Stattdessen sind andere äquivalente TESe oft geeigneter.

Um den ersten Problempunkt zu vermeiden, wollen wir daher sicher stellen, dass es keine unendlichen Reduktionen mit der Relation $\to_{\mathcal{R}}$ gibt. Im allgemeinen nennt man Relationen, die keine unendlich "absteigenden" Folgen von Elementen zulassen, *fundiert*. Dieser Begriff ist darüber hinaus sowohl für Beweise mit Induktion als auch für Terminierungsbeweise grundlegend.

Definition 3.3.8 (Fundierte Relationen) *Sei $\to$ eine Relation über einer Menge M. Die Relation $\to$ heißt* fundiert *gdw. es keine unendliche Folge $t_0, t_1, t_2, \ldots$ von Elementen aus M gibt mit $t_0 \to t_1 \to t_2 \to \ldots$*

Beispiel 3.3.9

- Die Relation $>_{\mathbb{N}}$ auf $\mathbb{N}$ ist fundiert, wobei "$>_{\mathbb{N}}$" die übliche "größer als"-Relation auf den natürlichen Zahlen ist.
- Die echte Teiltermrelation $\rhd$ auf $\mathcal{T}(\Sigma, \mathcal{V})$ ist fundiert.
- Die Relation $\to_{||}$ auf $\mathcal{T}(\Sigma, \mathcal{V})$ mit $t_1 \to_{||} t_2$ gdw. $|t_1| >_{\mathbb{N}} |t_2|$, ist fundiert. Hierbei bezeichnet $|t|$ die Anzahl der Symbole in t.
- Die Relation $<_{\mathbb{N}}$ auf $\mathbb{N}$ ist nicht fundiert.
- Für alle $c \in \mathbb{N}$ sei $<_c$ die Relation über $\mathbb{N}$ mit $n <_c m$ gdw. $c >_{\mathbb{N}} m >_{\mathbb{N}} n$. Dann ist $<_c$ fundiert.
- Die Relation $>_{\mathbb{Z}}$ auf $\mathbb{Z}$ ist nicht fundiert, wobei "$>_{\mathbb{Z}}$" die übliche "größer als"-Relation auf den ganzen Zahlen ist.
- Die Relation $>_{\mathbb{Q}}$ auf $\mathbb{Q}^+$ ist nicht fundiert, wobei "$>_{\mathbb{Q}}$" die übliche "größer als"-Relation auf den positiven rationalen Zahlen ist.
- Die Relation $>_{\mathbb{N}^*}$ auf $\mathbb{N}^*$ ist fundiert.

Unser Ziel ist also, TESe zu verwenden, bei denen die Ersetzungsrelation $\to_{\mathcal{R}}$ fundiert ist. Dann kann man jeden Term solange reduzieren, bis kein weiterer Reduktionsschritt mehr möglich ist. Das Resultat dieser Reduktion bezeichnet man als *Normalform*. (Analog geht man bei Programmiersprachen vor, bei denen Termersetzung zur *Auswertung* von Termen verwendet wird. Dort wird dann ein Ausdruck zu seiner Normalform ausgewertet, die sich als Ergebnis des Programms ergibt.)

Definition 3.3.10 (Normalform) *Sei $\to$ eine Relation über einer Menge M. Ein Objekt $q \in M$ heißt* Normalform *gdw. es kein Objekt $q' \in M$ mit $q \to q'$ gibt. Das Objekt q heißt* Normalform eines Objekts t *gdw. $t \to^* q$ und q Normalform ist. Falls die Normalform von t eindeutig ist, so bezeichnet $t{\downarrow}$ die Normalform von t.*

Die Relation $\to$ ist normalisierend*, wenn es zu jedem Objekt t (mindestens) eine Normalform gibt und $\to$ ist* eindeutig normalisierend*, wenn es zu jedem Objekt t* genau *eine Normalform gibt.*

Es ist leicht zu zeigen, dass aus der Fundiertheit die Normalisierung folgt, aber nicht umgekehrt.

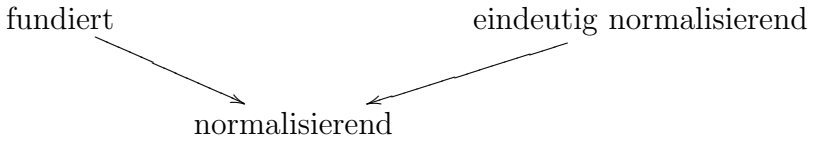
Lemma 3.3.11 (Fundiertheit impliziert Normalisierung) *Jede fundierte Relation ist normalisierend.*

Beweis. Sei $\to$ eine fundierte Relation über einer Menge M und sei $t \in M$. Wenn t keine Normalform hätte, dann gäbe es eine unendliche Folge

$$t \to t_1 \to t_2 \to \dots$$

im Widerspruch zur Fundiertheit. □

Die Umkehrung von Lemma 3.3.11 gilt hingegen nicht. Fundiertheit verlangt, dass *alle* absteigenden Folgen endlich sind. Normalisierung erfordert hingegen nur, dass es ausgehend von jedem Element mindestens *eine* abbrechende absteigende Folge gibt. Insgesamt gelten die folgenden Zusammenhänge (und alle anderen Zusammenhänge zwischen den drei Begriffen in dem Schaubild gelten nicht):



Bevor wir Beispiele für Relationen mit diesen drei Eigenschaften angeben, wollen wir definieren, wie man diese Begriffe auch für TESe verwenden kann.

Definition 3.3.12 (Terminierung von TESen) *Ein TES $\mathcal{R}$* terminiert *gdw. seine Ersetzungsrelation $\to_\mathcal{R}$ fundiert ist. Ein TES $\mathcal{R}$ ist* normalisierend *gdw. $\to_\mathcal{R}$ normalisierend ist, d.h., gdw. jeder Term eine Normalform bzgl. $\to_\mathcal{R}$ hat.*

In einem terminierenden TES hat also jeder Term *mindestens* eine Normalform. Bei jeder beliebigen Reduktion des Terms erreicht man nach endlich vielen Schritten eine solche Normalform.

Beispiel 3.3.13

(a) Das TES $\{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{f}(\mathsf{b})\}$ ist normalisierend, aber weder eindeutig normalisierend noch terminierend. Jeder Term hat nämlich Normalformen, wie man durch Induktion über den Termaufbau leicht zeigen kann (z.B., indem man alle Vorkommen von b durch a ersetzt). Der Term b hat z.B. die Normalformen $\mathsf{a}, \mathsf{f}(\mathsf{a})$, etc. Hingegen gibt es eine nicht-terminierende Reduktion $\mathsf{b} \to \mathsf{f}(\mathsf{b}) \to \mathsf{f}(\mathsf{f}(\mathsf{b})) \to \dots$.

(b) Das TES $\mathcal{R} = \{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{b}\}$ ist nicht terminierend, aber eindeutig normalisierend, d.h., hier hat jeder Term genau eine Normalform. Es gilt z.B. $\mathsf{b}{\downarrow_{\mathcal{R}}} = \mathsf{a}$.

(c) Das TES $\{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}\}$ terminiert, aber es gibt Terme mit mehreren Normalformen. So hat b sowohl die Normalform a als auch die Normalform c.

(d) Das TES $\{\mathsf{c} \to \mathsf{b}, \mathsf{a} \to \mathsf{b}\}$ terminiert und jeder Term hat genau eine Normalform. Dies sind die Arten von TESen, die wir im Folgenden zu konstruieren versuchen.

Man erkennt auch, dass die beiden Variablenbedingungen $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ und $l \notin \mathcal{V}$ für alle Regeln $l \to r \in \mathcal{R}$ notwendig sind, um terminierende TESe zu erhalten. Der Grund ist, dass man bei einer Variable $x \in \mathcal{V}(r) \setminus \mathcal{V}(l)$ mit $r|_\pi = x$ die folgende unendliche Reduktion bilden könnte, wobei $\sigma = \{x/l\}$ ist:

$$l \to r\sigma = r\sigma[l]_\pi \to r\sigma[r\sigma]_\pi = r\sigma[r\sigma[l]_\pi]_\pi \to r\sigma[r\sigma[r\sigma]_\pi]_\pi = \ldots$$

Analog würde man bei einer Regel $x \to r$ die folgende Reduktion erhalten, wobei $\sigma_1 = \{x/r\}$, $\sigma_2 = \{x/r\sigma_1\}$, etc.:

$$x \to r \to r\sigma_1 \to r\sigma_2 \to \ldots$$

Betrachten wir nun noch einmal unsere früheren Beispiele. In Bsp. 3.3.6 haben wir zur Untersuchung der Frage $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv_{\mathcal{E}} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ zunächst die *Normalformen* der beiden Terme $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x)$ und $\mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ gebildet. Anschließend haben wir überprüft, ob diese beiden Normalformen identisch waren. In diesem Beispiel war das der Fall, da beide Terme nur eine Normalform, nämlich $\mathsf{succ}(\mathsf{succ}(x))$ besitzen.

In Bsp. 3.3.7 hatten wir hingegen ein nicht-terminierendes TES $\mathcal{R}_1 = \{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}, \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{f}(\mathsf{a}))\}$. Bei der Untersuchung von $\mathsf{f}(\mathsf{a}) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{c})$ wollten wir ebenfalls Normalformen von $\mathsf{f}(\mathsf{a})$ und $\mathsf{f}(\mathsf{c})$ berechnen. Da $\mathsf{f}(\mathsf{a})$ aber keine Normalformen besitzt, scheitert dieser Ansatz. Das TES $\mathcal{R}_2 = \{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}, \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{f}(\mathsf{b}))\}$ ist im Gegensatz zu $\mathcal{R}_1$ wenigstens normalisierend, aber ebenfalls nicht terminierend. Hier besitzt $\mathsf{f}(\mathsf{a})$ zwar Normalformen $\mathsf{f}^2(\mathsf{c}), \mathsf{f}^3(\mathsf{c}), \ldots$, aber da $\mathcal{R}$ nicht terminiert, kann es passieren, dass die Reduktion von $\mathsf{f}(\mathsf{a})$ unendlich weiterläuft, ohne eine Normalform zu finden.

Die *Terminierung* stellt also sicher, dass wir beliebig reduzieren können und stets nach endlich vielen Schritten eine Normalform erhalten. Dies ist daher eine der Voraussetzungen, die wir benötigen, um TESe zur Lösung des Wortproblems zu verwenden. Man braucht die Terminierung auch, um zu untersuchen, ob ein TES eindeutige Normalformen hat, wie man in Kapitel 5 sehen wird. Darüber hinaus ist die Frage der Terminierung natürlich auch in anderen Bereichen überaus wichtig. Beispielsweise möchte man bei der Programmverifikation untersuchen, ob ein Programm bei allen Eingaben stets nach endlicher Zeit anhält. Die Terminierung ist darüber hinaus bei der automatisierten Programmverifikation wichtig, um Induktionsbeweise über Programme durchführen zu können (vgl. [Gie03]). Schließlich wird sich auch herausstellen, dass die Terminierung eine Voraussetzung für Vervollständigungsverfahren ist, die unvollständige Programme automatisch ergänzen können (Kapitel 6). In Kapitel 4 werden wir daher Verfahren vorstellen, die die Terminierung von TESen automatisch untersuchen.

Die TESe $\mathcal{R}_3$ und $\mathcal{R}_4$ aus Bsp. 3.3.7 und das TES aus Bsp. 3.3.13 (c) machen aber bereits deutlich, dass die Terminierung alleine noch nicht ausreicht, um unser skizziertes Termersetzungsverfahren für die Lösung des Wortproblems zu verwenden. Das TES $\{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}\}$ terminiert. Wenn man untersuchen möchte, ob in der zugrunde liegenden Menge von Gleichungen $\mathcal{E}$ die Aussage $\mathsf{a} \equiv_{\mathcal{E}} \mathsf{c}$ gilt, so würde man nun die Normalformen von a und c berechnen. Es gilt $\mathsf{a}{\downarrow_{\mathcal{R}}} = \mathsf{a}$ und $\mathsf{c}{\downarrow_{\mathcal{R}}} = \mathsf{c}$. Da die beiden resultierenden Terme aber nicht identisch sind, würde man nun fälschlicherweise schließen, dass $\mathsf{a} \not\equiv_{\mathcal{E}} \mathsf{c}$ gilt.

Das Problem hierbei ist, dass wir zur Untersuchung der Frage $s \equiv_{\mathcal{E}} t$ eigentlich untersuchen müssten, ob $s \leftrightarrow^*_{\mathcal{R}} t$ gilt. In Wirklichkeit untersuchen wir jedoch nur, ob es einen Term q gibt, so dass $s \to^*_{\mathcal{R}} q \leftarrow^*_{\mathcal{R}} t$ gilt. Man bezeichnet solche Terme dann als *zusammenführbar* (engl. joinable). Im allgemeinen folgt zwar aus der Zusammenführbarkeit von s und t, dass $s \leftrightarrow^*_{\mathcal{R}} t$ gilt, aber die Umkehrung gilt nicht. Dies wird an dem TES $\mathcal{R} = \{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}\}$ deutlich, bei dem $\mathsf{a} \leftrightarrow^*_{\mathcal{R}} \mathsf{c}$ gilt, obwohl die beiden Terme nicht zusammenführbar sind. Wir müssen uns also auf TESe beschränken, bei denen die Ersetzungsrelation $\to_{\mathcal{R}}$ diese Eigenschaft hat, die man allgemein für Relationen formulieren kann. Sie geht auf Church und Rosser zurück, die diese Eigenschaft für den Lambda-Kalkül untersuchten und bewiesen, vgl. [CR36, Gie05].

Definition 3.3.14 (Zusammenführbarkeit, Church-Rosser Eigenschaft) *Sei $\to$ eine Relation auf einer Menge M. Zwei Elemente $s, t \in M$ sind* zusammenführbar *(Schreibweise $s \downarrow t$) gdw. es ein $q \in M$ gibt mit $s \to^* q \leftarrow^* t$.*

Die Relation $\to$ hat die Church-Rosser *Eigenschaft gdw. für alle $s, t \in M$ gilt: Falls $s \leftrightarrow^* t$, dann sind s und t zusammenführbar, d.h., $s \downarrow t$.*

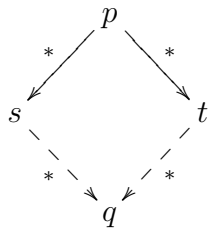
Man sagt, ein TES $\mathcal{R}$ hat die Church-Rosser *Eigenschaft gdw. seine Ersetzungsrelation $\to_{\mathcal{R}}$ die Church-Rosser Eigenschaft hat.*

Beispiel 3.3.15 Das TES $\mathcal{R} = \{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}\}$ aus Bsp. 3.3.13 (c) besitzt die Church-Rosser Eigenschaft nicht. So gilt zwar $\mathsf{a} \leftrightarrow^*_{\mathcal{R}} \mathsf{c}$, denn $\mathsf{a} \leftarrow_{\mathcal{R}} \mathsf{b} \to_{\mathcal{R}} \mathsf{c}$, aber a und c sind nicht zusammenführbar, denn beide sind Normalformen.

Das TES mit den Regeln (3.1) und (3.2) für die Addition besitzt hingegen die Church-Rosser Eigenschaft. Es gilt z.B. $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \leftrightarrow^*_{\mathcal{R}} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ und in der Tat sind die beiden Terme zusammenführbar (vgl. Bsp. 3.3.6).

Um TESe besser auf die Church-Rosser Eigenschaft untersuchen zu können, empfiehlt es sich, stattdessen die Eigenschaft der *Konfluenz* zu betrachten, die zur Church-Rosser Eigenschaft äquivalent ist. Auf einen Term p sind oft mehrere Auswertungsschritte anwendbar (d.h. wir haben eine Relation $\to$, bei der $p \to^* s$ und $p \to^* t$ für geeignete Objekte s, t, p gilt). Eine Relation $\to$ heißt *konfluent*, wenn es unerheblich ist, welchen der beiden Schritte man ausführt. Genauer bedeutet Konfluenz, dass sich die beiden "alternativen" Objekte s und t wieder zusammenführen lassen. Man kann dies durch folgendes Bild illustrieren: Wenn die durchgezogenen Pfeile existieren, folgt daraus die Existenz der gestrichelten Pfeile

und die Existenz eines geeigneten Objekts q.



Definition 3.3.16 (Konfluenz) *Eine Relation $\to$ über einer Menge M heißt* konfluent*, wenn für alle $s, t, p \in M$ gilt:*

Wenn $p \to^* s$ und $p \to^* t$,
dann existiert ein $q \in M$ mit $s \to^* q$ und $t \to^* q$.

Ein TES $\mathcal{R}$ heißt konfluent*, wenn die Ersetzungsrelation $\to_{\mathcal{R}}$ konfluent ist.*

Anschaulich bedeutet Konfluenz, dass zwei unterschiedliche "Wege" von p nach s und von p nach t immer zu einem *gemeinsamen* "Wegpunkt" q fortgesetzt werden können. Die Existenz mehrerer Wege von p aus repräsentiert einen Indeterminismus, denn man kann ausgehend von p sowohl nach s als auch nach t gelangen. Die Eigenschaft der Konfluenz gewährleistet dann, dass solche Indeterminismen beliebig aufgelöst werden können. Der folgende Satz zeigt, dass Konfluenz und die Church-Rosser Eigenschaft in der Tat äquivalent sind.

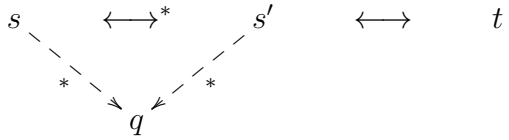
Satz 3.3.17 (Church-Rosser gdw. konfluent) *Für alle Relationen $\to$ gilt:*

$\to$ besitzt die Church-Rosser Eigenschaft gdw. $\to$ konfluent ist.

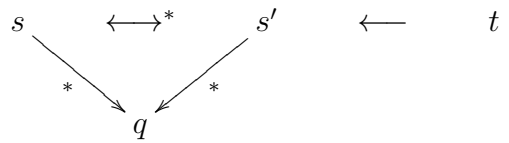
Beweis. Sei $\to$ zunächst eine Relation mit der Church-Rosser Eigenschaft. Für alle Elemente s, t, p mit $s \leftarrow^* p \to^* t$ gilt somit $s \leftrightarrow^* t$ und daher $s \downarrow t$, d.h., es existiert ein q mit $s \to^* q \leftarrow^* t$. Daher ist $\to$ dann konfluent.

Für die Rückrichtung sei $\to$ konfluent und sei $s \leftrightarrow^* t$. Wir müssen zeigen, dass dann s und t jeweils zusammenführbar sind. Wir beweisen die Aussage durch Induktion über die Anzahl der $\leftrightarrow$-Schritte zwischen s und t. Im Induktionsanfang ist $s = t$ und somit trivialerweise $s \downarrow t$.

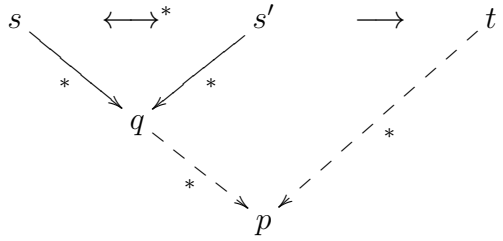
Im Induktionsschluss gilt $s \leftrightarrow^* s' \leftrightarrow t$ und aus der Induktionshypothese folgt $s \downarrow s'$. Es existiert also ein Element q mit $s \to^* q \leftarrow^* s'$.



Falls $s' \leftarrow t$ gilt, so folgt sofort $q \leftarrow^* s' \leftarrow t$, d.h., $s \downarrow t$.



Ansonsten gilt $s' \to t$. Wegen $q \leftarrow^* s' \to t$ folgt aus der Konfluenz von $\to$, dass $q \downarrow t$ gilt, d.h., es existiert ein p mit $q \to^* p \leftarrow^* t$. Damit ergibt sich $s \to^* q \to^* p \leftarrow^* t$, d.h., $s \downarrow t$.



□

Man erkennt, dass in dem Beispiel $\mathcal{R} = \{\mathsf{b} \to \mathsf{c}, \mathsf{b} \to \mathsf{a}\}$ das Problem daran liegt, dass der Term b mehrere Normalformen hat. In der Tat folgt aus der Konfluenz, dass jedes Objekt höchstens eine Normalform hat. Ist die Auswertung darüberhinaus normalisierend, so besitzt jeder Term genau eine Normalform. Dies macht deutlich, dass die Konfluenz generell für Programme wichtig ist. Man möchte, dass das Resultat der Auswertung eines Programms unabhängig von der verwendeten Auswertungsstrategie eindeutig ist (vgl. z.B. die Diskussion über "call-by-value" und "call-by-name"-Auswertung bei funktionalen Programmiersprachen [Gie05]). Sofern man weiß, dass das Programm terminiert (und damit normalisierend ist), ist die Konfluenz äquivalent zur Eindeutigkeit der Ergebnisse. In Kapitel 5 werden wir daher Verfahren untersuchen, um die Konfluenz automatisch nachzuweisen.

Lemma 3.3.18 (Konfluenz bedeutet eindeutige Normalformen)

(a) Wenn eine Relation $\to$ konfluent ist, dann hat jedes Objekt höchstens eine Normalform.

(b) Wenn $\to$ normalisierend und konfluent ist, dann hat jedes Element genau eine Normalform (d.h., $\to$ ist eindeutig normalisierend).

(c) Jede eindeutig normalisierende Relation ist konfluent.

Beweis.

(a) Sei t ein Objekt mit den Normalformen q_1 und q_2. Daraus folgt $t \to^* q_1$ und $t \to^* q_2$. Aufgrund der Konfluenz muss es also ein Objekt q geben mit $q_1 \to^* q \leftarrow^* q_2$. Da q_1 und q_2 aber Normalformen sind, folgt $q_1 = q = q_2$.

(b) Normalisierung bedeutet, dass jedes Objekt mindestens eine Normalform hat, und Konfluenz impliziert (nach (a)), dass jedes Objekt höchstens eine Normalform hat. Daher hat bei einer normalisierenden konfluenten Relation jedes Objekt also genau eine Normalform.

(c) Nun zeigen wir, dass aus der eindeutigen Normalisierung die Konfluenz folgt. Hierzu sei $s \leftarrow^* p \to^* t$. Wir müssen zeigen, dass dann s und t zusammenführbar sind. Seien q_1 und q_2 die (eindeutigen) Normalformen von s und t. Dann sind beide auch Normalformen von p und aufgrund der Eindeutigkeit folgt $q_1 = q_2$. Somit haben wir $s \to^* q_1 = q_2 \leftarrow^* t$. □

Ohne die Voraussetzung der Normalisierung folgt aus der Konfluenz natürlich nicht die eindeutige Normalisierung. So ist das TES mit der Regel $\mathsf{a} \to \mathsf{a}$ natürlich konfluent, aber es ist nicht normalisierend.

Bei normalisierenden und konfluenten Relationen $\to$ kann man anstelle der Überführbarkeit $s \leftrightarrow^* t$ die Zusammenführbarkeit $s \downarrow t$ untersuchen. Dies bedeutet insbesondere, dass man bei normalisierenden und konfluenten TESen $\mathcal{R}$ also nur die Normalformen von s und t berechnen und überprüfen muss, ob sie identisch sind, wenn man $s \leftrightarrow^*_{\mathcal{R}} t$ untersuchen will.

Satz 3.3.19 (Überführbarkeit und Zusammenführbarkeit) *Sei $\to$ eine normalisierende und konfluente Relation über M und seien $s, t \in M$. Dann gilt $s \leftrightarrow^* t$ gdw. $s{\downarrow} = t{\downarrow}$.*

Beweis. Aus $s{\downarrow} = t{\downarrow}$ folgt trivialerweise $s \leftrightarrow^* t$. Für die umgekehrte Richtung folgt aus $s \leftrightarrow^* t$ die Zusammenführbarkeit $s \downarrow t$, da die Konfluenz äquivalent zur Church-Rosser Eigenschaft ist (Satz 3.3.17). Es existiert daher ein Objekt $q \in M$ mit $s \to^* q \leftarrow^* t$. Da $\to$ normalisierend ist, existiert auch eine Normalform $q{\downarrow}$ von q. Somit ist $q{\downarrow}$ auch Normalform von s und t. Da aufgrund der Konfluenz Normalformen aber eindeutig sind (Lemma 3.3.18 (b)), folgt $s{\downarrow} = q{\downarrow} = t{\downarrow}$. □

Da aus der Fundiertheit die Normalisierung folgt (Lemma 3.3.11), bedeutet Lemma 3.3.18 (b), dass Normalformen bei fundierten und konfluenten Relationen existieren und eindeutig sind. Außerdem führt bei fundierten Relationen jede $\to$-Folge zu dieser Normalform. Satz 3.3.19 bedeutet dann, dass man bei terminierenden und konfluenten TESen $s \leftrightarrow^*_{\mathcal{R}} t$ entscheiden kann, indem man s und t zu ihrer Normalform reduziert. Unser Ziel ist daher, TESe zu konstruieren, die sowohl terminieren als auch konfluent sind. Deshalb führen wir für solche TESe einen eigenen Begriff ein.

Definition 3.3.20 (Konvergente TESe) *Ein TES ist* konvergent *(oder* kanonisch*), wenn es terminiert und konfluent ist.*

Beispiel 3.3.21 Bei konvergenten TESen $\mathcal{R}$ kann man $\downarrow_{\mathcal{R}}$ als *Interpreter* für das durch die Gleichungen axiomatisierte Programm verwenden. Für jeden Grundterm t erhält man durch $t{\downarrow_{\mathcal{R}}}$ den "Wert" von t. Für das Programm aus den beiden plus-Gleichungen (3.1) und (3.2) kann man also nun $\mathsf{plus}(2,1)$ auswerten lassen. Hierzu werden 2 und 1 in die "rechnerinterne" Darstellung $\mathsf{succ}^2(\mathcal{O})$ und $\mathsf{succ}(\mathcal{O})$ übersetzt und dann wird für den Term $t = \mathsf{plus}(\mathsf{succ}^2(\mathcal{O}), \mathsf{succ}(\mathcal{O}))$ die Normalform $t{\downarrow_{\mathcal{R}}}$ berechnet, vgl. Bsp. 3.3.2. Bei konvergenten TESen ist es dabei unerheblich, welche Regel angewendet wird, falls mehrere anwendbar sind, und es ist garantiert, dass man nach endlich vielen Schritten die Normalform erreicht. In unserem Beispiel erhält man auf diese Weise das Ergebnis $\mathsf{succ}^3(\mathcal{O})$, d.h., 3.

Das obige Beispiel zeigt, dass man mit konvergenten TESen "rechnen" kann. Unser Ziel ist jedoch ehrgeiziger, denn wir wollen auch Beweise führen. In der Tat lässt sich bei konvergenten TESen unser Beweisverfahren für das Wortproblem nun wie gewünscht durchführen, d.h., konvergente TES sind nicht nur "Interpreter", sondern auch "automatische Beweiser"

für Gleichungen. Um $s \equiv_{\mathcal{E}} t$ zu untersuchen, benötigt man ein konvergentes TES $\mathcal{R}$, das äquivalent zu $\mathcal{E}$ ist. Dann kann man das Wortproblem durch folgenden Algorithmus lösen. (Wie immer gehen wir hierbei wieder von endlichen Gleichungssystemen $\mathcal{E}$ und TESen $\mathcal{R}$ aus.)

Algorithmus WORTPROBLEM$(\mathcal{R}, s, t)$
Eingabe: Ein konvergentes TES $\mathcal{R}$ über Σ und $\mathcal{V}$ (äquivalent zu $\mathcal{E}$) und $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$.
Ausgabe: "*True*", falls $s \equiv_{\mathcal{E}} t$, und sonst "*False*".

1. Reduziere s und t auf beliebige Weise mit $\to_{\mathcal{R}}$ solange wie möglich.
   Auf diese Weise entstehen die Normalformen $s{\downarrow_{\mathcal{R}}}$ und $t{\downarrow_{\mathcal{R}}}$.
2. Falls $s{\downarrow_{\mathcal{R}}} = t{\downarrow_{\mathcal{R}}}$, dann gib "*True*" aus.
   Sonst gib "*False*" aus.

Satz 3.3.22 (Überprüfung des Wortproblems mit TESen)

(a) Der Algorithmus WORTPROBLEM *terminiert.*

(b) Falls $\mathcal{R}$ äquivalent zum Gleichungssystem $\mathcal{E}$ ist, so ist der Algorithmus WORTPROBLEM *korrekt.*

(c) Existiert zu einem Gleichungssystem ein äquivalentes konvergentes TES, so ist das Wortproblem über diesem Gleichungssystem entscheidbar.

Beweis.

(a) Die Terminierung folgt sofort aus der Terminierung von $\mathcal{R}$, da jede Reduktionsfolge von s und t endlich ist.

(b) Die Korrektheit des Algorithmus folgt aus Satz 3.3.19, da aus der Fundiertheit von $\to_{\mathcal{R}}$ die Normalisierung von $\to_{\mathcal{R}}$ folgt (Lemma 3.3.11) und demnach $s{\downarrow_{\mathcal{R}}} = t{\downarrow_{\mathcal{R}}}$ genau dann gilt, wenn $s \leftrightarrow^*_{\mathcal{R}} t$ gilt. Dies genügt für die Korrektheit, denn $\mathcal{R}$ ist nach Voraussetzung äquivalent zu $\mathcal{E}$ (d.h., $s \leftrightarrow^*_{\mathcal{R}} t$ gdw. $s \leftrightarrow^*_{\mathcal{E}} t$ gdw. $s \equiv_{\mathcal{E}} t$ nach dem Satz von Birkhoff, Satz 3.1.14).

(c) Die Entscheidbarkeit folgt sofort aus (b), da wir hierzu den Algorithmus WORTPROBLEM als Entscheidungsverfahren verwenden können. Hierzu muss man sich davon überzeugen, dass "Termersetzung", d.h., das Reduzieren mit Regeln, effektiv (automatisch) durchführbar ist. Der Grund ist, dass man effektiv feststellen kann, ob ein Term t reduzierbar ist. Hierzu muss man die (endlich vielen) Teilterme von t und die endlich vielen linken Seiten von Regeln aus $\mathcal{R}$ untersuchen und jeweils überprüfen, ob der Term aus der Regel den Teilterm von t matcht. Einen Algorithmus, um Matching automatisch durchzuführen und entsprechende Matcher zu berechnen, werden wir in Kapitel 5 kennen lernen. Anschließend muss man den Teilterm durch die entsprechend (durch den Matcher) instantiierte rechte Seite ersetzen. □

Beispiel 3.3.23 Wir betrachten wieder die Gleichungen $\mathcal{E}$ zur Axiomatisierung von Gruppen, d.h., $\mathcal{E} = \{\mathsf{f}(x,\mathsf{f}(y,z)) \equiv \mathsf{f}(\mathsf{f}(x,y),z), \mathsf{f}(x,\mathsf{e}) \equiv x, \mathsf{f}(x,\mathsf{i}(x)) \equiv \mathsf{e}\}$. Bei Ersetzung des Gleichheitszeichens $\equiv$ durch einen Pfeil $\to$ erhält man ein zu $\mathcal{E}$ äquivalentes TES:

$$\begin{array}{rcll} \mathsf{f}(x,\mathsf{f}(y,z)) & \to & \mathsf{f}(\mathsf{f}(x,y),z) & (3.3) \\ \mathsf{f}(x,\mathsf{e}) & \to & x & (3.4) \\ \mathsf{f}(x,\mathsf{i}(x)) & \to & \mathsf{e} & (3.5) \end{array}$$

Das obige TES terminiert, d.h., jeder Term besitzt (mindestens) eine Normalform. Wenn wir wieder untersuchen wollen, ob $\mathsf{i}(\mathsf{i}(n)) \equiv_{\mathcal{E}} n$ gilt, können wir den Algorithmus WORTPROBLEM aber nicht benutzen. Das Problem ist, dass sowohl $\mathsf{i}(\mathsf{i}(n))$ als auch n bereits Normalformen sind. Diese sind nicht identisch, obwohl wir bereits in Bsp. 3.1.15 gezeigt hatten, dass $\mathsf{i}(\mathsf{i}(n)) \equiv_{\mathcal{E}} n$ eine wahre Aussage ist.

Das Problem liegt daran, dass das obige TES nicht konfluent ist. Beispielsweise gilt

$$\begin{array}{l} \underline{\mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(y)))} \to \mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) \quad \text{und} \\ \mathsf{f}(x,\underline{\mathsf{f}(y,\mathsf{i}(y))}) \to \underline{\mathsf{f}(x,\mathsf{e})} \to x, \end{array}$$

d.h., $\mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) \leftrightarrow^* x$. Die beiden Terme sind aber nicht zusammenführbar, d.h., $\mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) \not\downarrow x$. Wenn man daran interessiert ist, welche Aussagen aus einem Gleichungssystem folgen, kann man also im allgemeinen nicht einfach dieses Gleichungssystem als TES verwenden, denn typischerweise ist solch ein TES nicht konvergent. In unserem Beispiel muss man das obige TES um einige weitere Regeln ergänzen, so dass das TES weiterhin äquivalent zu $\mathcal{E}$ und terminierend bleibt, aber zusätzlich auch konfluent wird. Wenn man beispielsweise die Regel $\mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) \to x$ hinzufügt, sind die beiden Terme natürlich zusammenführbar. Insgesamt muss man das obige TES wie folgt *vervollständigen:*

$$\begin{array}{rcll} \mathsf{f}(x,\mathsf{f}(y,z)) & \to & \mathsf{f}(\mathsf{f}(x,y),z) & (3.3) \\ \mathsf{f}(x,\mathsf{e}) & \to & x & (3.4) \\ \mathsf{f}(x,\mathsf{i}(x)) & \to & \mathsf{e} & (3.5) \\ \mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) & \to & x & (*1) \\ \mathsf{i}(\mathsf{e}) & \to & \mathsf{e} & (*2) \end{array} \qquad \begin{array}{rcll} \mathsf{f}(\mathsf{e},x) & \to & x & (*3) \\ \mathsf{i}(\mathsf{i}(x)) & \to & x & (*4) \\ \mathsf{f}(\mathsf{i}(x),x) & \to & \mathsf{e} & (*5) \\ \mathsf{f}(\mathsf{f}(x,\mathsf{i}(y)),y) & \to & x & (*6) \\ \mathsf{i}(\mathsf{f}(x,y)) & \to & \mathsf{f}(\mathsf{i}(y),\mathsf{i}(x)) & (*7) \end{array}$$

Dieses TES $\mathcal{R}$ ist konvergent und äquivalent zu $\mathcal{E}$. Man kann daher mit diesem TES jede (Gleichheits-)aussage über Gruppen automatisch entscheiden. Für die Aussage $\mathsf{i}(\mathsf{i}(n)) \equiv_{\mathcal{E}} n$ ist dies natürlich trivial, denn mit Regel $(*4)$ gilt $\mathsf{i}(\mathsf{i}(n)) \to_{\mathcal{R}} n$.

Der Nachweis von $\mathsf{i}(\mathsf{f}(\mathsf{i}(u),\mathsf{f}(v,u))) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{i}(u),\mathsf{f}(\mathsf{i}(v),u))$ ist hingegen aufwendiger. Hierzu berechnet man gemäß des Algorithmus WORTPROBLEM zuerst die Normalform von $\mathsf{i}(\mathsf{f}(\mathsf{i}(u),\mathsf{f}(v,u)))$. Hierbei darf man beliebige Reduktionen anwenden.

$$\begin{array}{lll} \underline{\mathsf{i}(\mathsf{f}(\mathsf{i}(u),\mathsf{f}(v,u)))} & \mathsf{i}(\mathsf{f}(x,y)) \to \mathsf{f}(\mathsf{i}(y),\mathsf{i}(x)) \quad (*7) & \sigma = \{x/\mathsf{i}(u), y/\mathsf{f}(v,u)\} \\ \mathsf{f}(\underline{\mathsf{i}(\mathsf{f}(v,u))},\mathsf{i}^2(u)) & \mathsf{i}(\mathsf{f}(x,y)) \to \mathsf{f}(\mathsf{i}(y),\mathsf{i}(x)) \quad (*7) & \sigma = \{x/v, y/u\} \\ \mathsf{f}(\underline{\mathsf{f}(\mathsf{i}(u),\mathsf{i}(v))},\underline{\mathsf{i}^2(u)}) & \mathsf{i}(\mathsf{i}(x)) \to x \quad (*4) & \sigma = \{x/u\} \\ \mathsf{f}(\mathsf{f}(\mathsf{i}(u),\mathsf{i}(v)),u) & & \end{array}$$

Anschließend berechnet man die Normalform von $\mathsf{f}(\mathsf{i}(u),\mathsf{f}(\mathsf{i}(v),u))$:

$$\begin{array}{ll} \underline{\mathsf{f}(\mathsf{i}(u),\mathsf{f}(\mathsf{i}(v),u))} & \mathsf{f}(x,\mathsf{f}(y,z)) \to \mathsf{f}(\mathsf{f}(x,y),z) \quad (3.3) \qquad \sigma = \{x/\mathsf{i}(u), y/\mathsf{i}(v), z/u\} \\ \mathsf{f}(\mathsf{f}(\mathsf{i}(u),\mathsf{i}(v)),u) & \end{array}$$

Nun überprüft man, ob die beiden Normalformen identisch sind. Da dies hier der Fall ist, ist damit $\mathsf{i}(\mathsf{f}(\mathsf{i}(u), \mathsf{f}(v, u))) \equiv_{\mathcal{E}} \mathsf{f}(\mathsf{i}(u), \mathsf{f}(\mathsf{i}(v), u))$ bewiesen.

Bei Verwendung eines konvergenten Termersetzungssystems ist der Aufwand zur Untersuchung des Wortproblems nicht mehr notwendigerweise exponentiell (wie in Abschnitt 3.1), denn der Verzweigungsgrad bei der Reduktion ist jetzt 1. Indeterminismen dürfen nun willkürlich aufgelöst werden, ohne die Beweisbarkeit zu verlieren. Es handelt sich jetzt auch nicht mehr um ein *Such*verfahren, sondern die Gültigkeit einer Gleichung $s \equiv_{\mathcal{E}} t$ wird hier genauso *ausgerechnet*, wie sich $\mathsf{plus}(2, 1)$ ausrechnen lässt. Die Voraussetzung dabei ist, dass $\mathcal{R}$ äquivalent zu dem Gleichungssystem $\mathcal{E}$ und konvergent (d.h., terminierend und konfluent) ist. Im Unterschied zum Verfahren von Abschnitt 3.1 handelt es sich hier auch nicht mehr um ein *Semi-Entscheidungsverfahren*. Der Algorithmus WORTPROBLEM terminiert stets, d.h., er kann die Gültigkeit einer Gleichung *entscheiden*.

Im Folgenden wird es daher darum gehen, zu einem Gleichungssystem $\mathcal{E}$ ein äquivalentes und konvergentes TES zu konstruieren, um ein Entscheidungsverfahren für das Wortproblem über $\mathcal{E}$ zu erhalten. Da das Wortproblem für beliebige Gleichungssysteme $\mathcal{E}$ im allgemeinen jedoch *unentscheidbar* ist (vgl. [BN98, Ex. 4.1.3, 4.1.4], [Ave95, Abschnitt 2.4]), wird dies nicht immer gelingen. Jedoch ist man für viele praktisch relevante Gleichungsmengen erfolgreich, wie sich im Folgenden herausstellen wird. Zur Lösung dieser Aufgabe gehen wir im Prinzip auf folgende Art und Weise vor:

1. Konstruiere zu der gegebenen Gleichungsmenge $\mathcal{E}$ ein äquivalentes TES $\mathcal{R}$ (z.B. durch Richten der Gleichungen in $\mathcal{E}$).

2. Überprüfe, ob $\mathcal{R}$ terminiert. Wenn dies nicht gezeigt werden kann, gibt man mit Fehlschlag auf.

3. Überprüfe, ob $\mathcal{R}$ konfluent ist. Falls ja, so ist $\mathcal{R}$ konvergent und zu $\mathcal{E}$ äquivalent. Wir haben damit ein Entscheidungsverfahren für das Wortproblem über $\mathcal{E}$ konstruiert und können das Vorgehen mit Erfolg beenden.

4. Falls nein, so führen wir einen *Vervollständigungsschritt* aus, indem wir $\mathcal{R}$ um neue Regeln ergänzen. Hierbei muss sicher gestellt sein, dass dadurch die Korrektheit von $\mathcal{R}$ erhalten bleibt (d.h., man darf $\mathcal{R}$ nur um solche Regeln $l \to r$ ergänzen, für die $l \equiv_{\mathcal{E}} r$ gilt). Nun geht man zurück zu Schritt 2.

Das Vorgehen lässt sich durch das Flußdiagramm in Abb. 3.1 verdeutlichen. Es stellen sich somit folgende Fragen:

I. Wie kann man zu einem TES herausfinden, ob es terminiert? (Kapitel 4)

II. Wie kann man zu einem (terminierenden) TES herausfinden, ob es konfluent ist? (Kapitel 5)

III. Wie kann man ein terminierendes TES vervollständigen, um ein konfluentes (und äquivalentes) TES zu erhalten? (Kapitel 6)

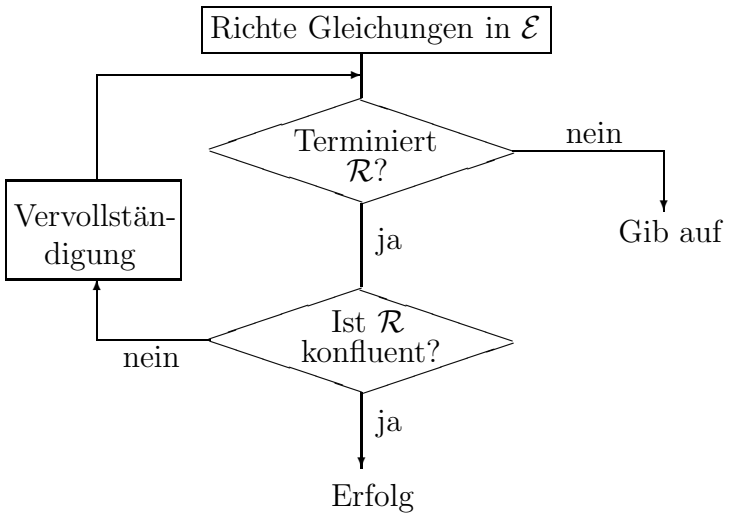


Abbildung 3.1: Vorgehen zur Konstruktion äquivalenter konvergenter TESe

Diese drei Fragen werden wir in den nächsten drei Kapiteln untersuchen und automatische Verfahren kennen lernen, um diese Fragestellungen zu bearbeiten. Diese Punkte haben auch generelle Bedeutung bei der Analyse und Verifikation von Programmen, da dort die Frage des Terminierungsnachweises, der Eindeutigkeit von Programmen und der automatischen Vervollständigung unvollständiger Programme ebenfalls von großer Wichtigkeit sind.

Die Frage der Terminierung (I.) ist unentscheidbar (dies entspricht der Unentscheidbarkeit des Halteproblems). Daher werden wir uns in Kapitel 4 mit hinreichenden Kriterien begnügen müssen, um die Terminierung von TESen zu erkennen. Hingegen ist es für terminierende TESe entscheidbar, ob sie konfluent sind. Die Frage II. kann daher immer automatisch beantwortet werden und wir werden in Kapitel 5 hierzu ein Verfahren kennen lernen. Wenn durch das Richten der Gleichungen ein zwar terminierendes, aber nicht konfluentes TES $\mathcal{R}_0$ entsteht, so fügt man in einem Vervollständigungsverfahren (Kapitel 6) neue Regeln hinzu, die die Korrektheit (und damit die Äquivalenz) des TES erhalten. Dies führt zu einem TES $\mathcal{R}_1$, das nun wieder auf Terminierung und Konfluenz überprüft wird. Falls $\mathcal{R}_1$ wieder terminiert, aber nicht konfluent ist, so fügt man erneut weitere Regeln hinzu und erhält ein neues TES $\mathcal{R}_2$, etc.

Man erhält so eine Folge $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$ von TESen, die alle zu $\mathcal{E}$ äquivalent sind. Es gibt nun 3 Möglichkeiten für diesen Prozess:

1. Man könnte nach einigen Iterationen ein TES $\mathcal{R}_n$ erhalten, das sowohl terminiert als auch konfluent ist. Dann endet das Verfahren mit Erfolg und man hat ein konvergentes und zu $\mathcal{E}$ äquivalentes TES gefunden.

2. Es könnte sein, dass man bei einem TES $\mathcal{R}_n$ in der Folge die Terminierung nicht nachweisen kann. (Dies kann daran liegen, dass es wirklich nicht terminiert oder dass unsere hinreichenden Kriterien nur die Terminierung nicht nachweisen können.) Nun bricht man mit Misserfolg ab. Immerhin erkennt man in diesem Fall den Misserfolg und könnte z.B. mit einem verbesserten Terminierungstest einen neuen Vervollständigungsversuch starten.

3. Schließlich kann es auch passieren, dass zwar alle $\mathcal{R}_i$ in der Folge terminieren, man aber kein konfluentes TES erreicht. Dann terminiert das Vervollständigungsverfahren nicht und in diesem Fall erkennen wir daher die Erfolglosigkeit des Verfahrens auch nicht.

Kapitel 4

Terminierung von Termersetzungssystemen

Im vorigen Kapitel wurde ein Entscheidungsverfahren (der Algorithmus WORTPROBLEM) für das Wortproblem über einem Gleichungssystem $\mathcal{E}$ vorgestellt. Dieses Verfahren verwendet ein zu $\mathcal{E}$ äquivalentes Termersetzungssystem $\mathcal{R}$, das *konvergent* ist, d.h., es muss terminieren und konfluent sein. Wie in Abschnitt 3.3 bereits skizziert, benötigt man daher insbesondere ein Verfahren, um (automatisch) zu überprüfen, ob ein gegebenes TES terminiert. Außerdem wird sich herausstellen, dass die Terminierung eines TES auch Voraussetzung für das Verfahren zur Überprüfung der Konfluenz in Kapitel 5 ist.

Die Fragestellung der Terminierung ist darüber hinaus natürlich generell bei der Softwareentwicklung von großem Interesse. Von einem korrekten Programm wird (oftmals) erwartet, dass es sein Resultat in endlicher Zeit berechnet und nicht in eine "Endlosschleife" gerät. Die Terminierungsanalyse ist daher ein wichtiger Bestandteil der Programmverifikation. In diesem Kapitel werden daher Techniken vorgestellt, um die Terminierung eines TES automatisch nachzuweisen. Diese Techniken lassen sich dann auch für die Terminierungsanalyse von Programmen in anderen Programmiersprachen verwenden, vgl. [Gie03].

Die Terminierung eines Programms oder eines TES hängt direkt mit dem Begriff der *fundierten* Relation zusammen, denn ein TES $\mathcal{R}$ terminiert gdw. seine Ersetzungsrelation $\to_{\mathcal{R}}$ fundiert ist. In Abschnitt 4.1 stellen wir ein grundlegendes Resultat über fundierte Relationen vor, das im Folgenden verwendet wird. Es zeigt den Zusammenhang zwischen Terminierung und Induktion, wodurch deutlich wird, dass ein Verfahren zur automatischen Terminierungsanalyse auch benötigt wird, um rechnergestützte Induktionsbeweise durchzuführen [Gie03].

Anschließend betrachten wir in Abschnitt 4.2 ein erstes Verfahren zur Terminierungsüberprüfung bei solchen TESen, bei denen auf den rechten Seiten der Regeln keine Variablen auftreten. Während hier die Frage der Terminierung entscheidbar ist, lässt sich im allgemeinen kein Verfahren angeben, das die Terminierung aller TESe entscheiden kann (dies beruht auf der Unentscheidbarkeit des Halteproblems). Im allgemeinen werden wir uns also mit einem hinreichenden und unvollständigen Verfahren zufrieden geben müssen.

Hierzu entwickeln wir in Abschnitt 4.3 einen Ansatz für (automatische) Terminierungsbeweise, der darauf beruht, die linken und rechten Seiten der Regeln mit Hilfe bestimmter Relationen zu vergleichen. Eine besonders gut dafür geeignete Klasse von Relationen, die

sich automatisch generieren lassen, sind die lexikographische und die rekursive Pfadordnung, die in Abschnitt 4.4 vorgestellt werden.

4.1 Noethersche Induktion

Wie erläutert, bedeutet der Terminierungsnachweis, dass man zeigen muss, ob eine bestimmte Relation fundiert ist, d.h., ob es keine unendlich absteigenden Folgen mit dieser Relation gibt. Die Frage, ob eine Relation fundiert ist, stellt sich nicht nur für Terminierungsuntersuchungen, sondern sie ist auch nötig, wenn man *Induktionsbeweise* anhand dieser Relation durchführen will.

Wir haben bereits strukturelle Induktion für verschiedene Datenstrukturen (wie natürliche Zahlen, Terme, Stellen, etc.) kennen gelernt. Es stellt sich heraus, dass all diese Beweisverfahren Spezialfälle eines noch allgemeineren Beweisprinzips (der sogenannten "*noetherschen Induktion*") sind.

Das Induktionsprinzip für natürliche Zahlen besagt, dass es ausreicht, $\varphi(0)$ und $\forall y \in \mathbb{N}.\ \varphi(y) \Rightarrow \varphi(y+1)$ zu zeigen, um $\forall x \in \mathbb{N}.\ \varphi(x)$ zu beweisen. Diese Form der Induktion wird auch als *Peano-Induktion* bezeichnet. Analoge Induktionsprinzipien haben wir für die Datenstruktur der Terme und die Datenstruktur der Stellen betrachtet. Im Folgenden verwenden wir den Quantor "$\forall$" und die Implikation "$\Rightarrow$" als Kurzschreibweise in der Metasprache, d.h., in der Sprache, in der wir *über* Formeln oder andere Konzepte der Objektsprache reden. Die bislang verwendeten strukturellen Induktionsprinzipien lauten dann wie folgt:

$$\varphi(0) \wedge (\forall y \in \mathbb{N}.\ \ \varphi(y) \Rightarrow \varphi(y+1)) \ \Rightarrow\ \forall x \in \mathbb{N}.\ \ \varphi(x) \tag{4.1}$$

$$\begin{array}{l}(\forall x \in \mathcal{V}.\ \ \varphi(x)) \ \wedge \\ (\forall t_1,..,t_n \in \mathcal{T}(\Sigma,\mathcal{V}).\ \forall f \in \Sigma.\ \ \varphi(t_1) \wedge .. \wedge \varphi(t_n) \Rightarrow \varphi(f(t_1,..,t_n)))\end{array} \ \Rightarrow\ \forall t \in \mathcal{T}(\Sigma,\mathcal{V}).\ \varphi(t) \tag{4.2}$$

$$\varphi(\epsilon) \wedge (\forall \pi' \in \mathbb{N}^*.\ \forall i \in \mathbb{N}.\ \ \varphi(\pi') \Rightarrow \varphi(i\pi')) \ \Rightarrow\ \forall \pi \in \mathbb{N}^*.\ \ \varphi(\pi) \tag{4.3}$$

$$\varphi(\epsilon) \wedge (\forall \pi' \in \mathbb{N}^*.\ \forall i \in \mathbb{N}.\ \ \varphi(\pi') \Rightarrow \varphi(\pi' i)) \ \Rightarrow\ \forall \pi \in \mathbb{N}^*.\ \ \varphi(\pi) \tag{4.4}$$

Diese Induktionsprinzipien lassen sich nun verallgemeinern. Zum einen werden wir nicht nur über natürliche Zahlen, Terme oder Stellen induzieren, sondern über beliebige Mengen M. Außerdem werden wir auch weitere Arten des Induktionsschritts einführen. Im Peano-Induktionsprinzip wird der Induktionsschritt von y auf $y+1$ durchgeführt. Mit anderen Worten, wenn man im Induktionsschluss die Aussage für eine Zahl $y+1$ zeigen will, darf man als Induktionshypothese voraussetzen, dass die Aussage bereits für den direkten Vorgänger y gilt.

Wir verallgemeinern dieses Induktionsprinzip jetzt wie folgt: Wenn man im Induktionsschluss die Aussage für ein Objekt $m \in M$ zeigen will, so darf man als Induktionshypothese voraussetzen, dass die Aussage bereits für alle solchen Objekte $k \in M$ gilt, die *kleiner* als m sind. Hierbei muss man natürlich festlegen, was es bedeuten soll, dass manche Objekte aus M "*kleiner*" als andere Objekte aus M sind. Im folgenden Induktionsprinzip darf man zum Vergleich von Objekten aus M eine beliebige Relation $\succ$ auf M verwenden; die einzige Bedingung, die an $\succ$ gestellt wird, ist, dass sie *fundiert* ist. Die folgende Definition führt dieses Induktionsprinzip ein, das nach der Logikerin Emmy Noether benannt ist.

Definition 4.1.1 (Noethersches Induktionsprinzip) *Sei $\succ$ eine fundierte Relation über einer Menge M und sei $\varphi(m)$ eine Aussage über Objekte m aus M. Für alle $m \in M$ gelte der folgende Sachverhalt:*

$$\begin{array}{l} \textit{Wenn } \varphi(k) \textit{ für alle } k \in M \textit{ mit } m \succ k \textit{ gilt,} \\ \textit{so gilt auch } \varphi(m). \end{array} \tag{4.5}$$

Dann gilt die Aussage $\varphi(n)$ für alle $n \in M$.

Mit der obigen Kurzschreibweise lässt sich das noethersche Induktionsprinzip wie folgt ausdrücken:

$$(\ \forall m \in M.\ (\ \forall k \in M.\ m \succ k\ \Rightarrow\ \varphi(k)\ )\ \Rightarrow\ \varphi(m)\ )\quad \Rightarrow\quad \forall n \in M.\ \varphi(n)$$

Mit der noetherschen Induktion reicht es also, die noethersche Induktionsformel

$$\forall m \in M.\ (\ \forall k \in M.\ m \succ k\ \Rightarrow\ \varphi(k)\ )\ \Rightarrow\ \varphi(m) \tag{4.6}$$

zu zeigen, wenn $\forall n \in M.\ \varphi(n)$ bewiesen werden soll.

Man erkennt, dass das Peano-Induktionsprinzip (4.1) ein Spezialfall der noetherschen Induktion ist. Hierbei wählt man als M die natürlichen Zahlen $\mathbb{N}$ und als $\succ$ wählt man die Relation, bei der $m \succ k$ für zwei natürliche Zahlen m und k gilt, falls $m = k + 1$ ist. Das Peano-Induktionsprinzip führt dabei eine Fallunterscheidung nach $m = 0$ und $m \neq 0$ durch, was zum Induktionsanfang und zum Induktionsschluss führt. Im noetherschen Induktionsprinzip verschmelzen Induktionsanfang und Induktionsschritt hingegen zu einer Formel (4.6). Je nach Wahl der Relation $\succ$ kann es aus beweistechnischer Sicht günstig sein, verschiedene Fallunterscheidungen durchzuführen. Dies kann dann zu mehreren Induktionsformeln führen. Im allgemeinen bezeichnen wir jene Fälle als *Induktionsanfang*, bei denen nur Objekte m betrachtet werden, zu denen es keine kleineren Objekte k mit $m \succ k$ gibt. Die anderen Fälle entsprechen dem *Induktionsschluss*. Wie üblich wird die Teilformel $\forall k \in M.\ m \succ k\ \Rightarrow\ \varphi(k)$ aus (4.6) als *Induktionshypothese* bezeichnet.

Bei der Induktion über Termen (4.2) wird als fundierte Relation die direkte Teiltermrelation verwendet. Selbstverständlich wäre es auch möglich, stattdessen die echte Teiltermrelation $\rhd$ zu benutzen, denn diese ist ebenfalls fundiert. Dann hätte man im Induktionsschluss beim Beweis von $\varphi(t)$ die Aussage $\varphi(q)$ für alle echten Teilterme q von t zur Verfügung.

Bei den Induktionen über Stellen werden zwei verschiedene fundierte Relationen verwendet. Im Induktionsprinzip (4.3) benutzt man eine Relation mit $\pi_1 \succ \pi_2$ gdw. $\pi_1 = i\pi_2$ für ein $i \in \mathbb{N}$. Im anderen Induktionsprinzip (4.4) setzt man hingegen $\pi_1 \succ \pi_2$ gdw. $\pi_1 = \pi_2 i$ für ein $i \in \mathbb{N}$. Diese zweite Relation bedeutet, dass $\pi_1 \succ \pi_2$ gilt gdw. die Stelle π_2 direkt oberhalb der Stelle π_1 liegt. Natürlich wäre es auch möglich, stattdessen die Relation $>_{\mathbb{N}^*}$ zu verwenden, die ebenfalls fundiert ist. Dann hätte man im Induktionsschluss beim Beweis von $\varphi(\pi_1)$ die Aussage $\varphi(\pi_2)$ für alle Stellen π_2 echt oberhalb von π_1 zur Verfügung.

Satz 4.1.2 (Korrektheit der Noetherschen Induktion) *Das noethersche Induktionsprinzip ist korrekt.*

Beweis. Wir nehmen an, die Induktionsformel (4.5) bzw. (4.6) gälte für alle $m \in M$, es gäbe aber ein $n_0 \in M$, so dass $\varphi(n_0)$ nicht gilt. Da (4.5) bzw. (4.6) auch für n_0 gilt, kann damit $\varphi(k)$ nicht für alle $k \in M$ mit $n_0 \succ k$ zutreffen. Es gibt also ein $n_1 \in M$ mit $n_0 \succ n_1$, so dass $\varphi(n_1)$ ebenfalls nicht gilt. Analog konstruiert man dann auch ein $n_2 \in M$ mit $n_1 \succ n_2$, so dass $\varphi(n_2)$ auch nicht gilt, etc. Insgesamt[1] erhält man also eine unendlich absteigende Folge

$$n_0 \succ n_1 \succ n_2 \succ \ldots$$

Dies widerspricht aber der Fundiertheit von $\succ$. □

Der nächste wichtige Satz, den wir im Folgenden benötigen, zeigt eine Anwendung der noetherschen Induktion außerhalb der natürlichen Zahlen, Terme oder Stellen.

Satz 4.1.3 (Lemma von König) *Ein Baum mit endlichem Verzweigungsgrad, in dem jeder Pfad endlich ist, besitzt nur endlich viele Knoten.*

Beweis. Wir betrachten einen Baum B mit endlichem Verzweigungsgrad, in dem jeder Pfad endlich ist. Sei M die Menge aller Knoten des Baums und für alle $m \in M$ sei B_m der Teilbaum, der vom Knoten m aufgespannt wird. Weiter sei $\succ$ eine Relation auf Knoten mit $m \succ k$ gdw. k direktes Kind des Knotens m ist. Da jeder Pfad in dem Baum endlich ist, ist $\succ$ fundiert. Wir zeigen für jeden Knoten $n \in M$ die folgende Aussage $\varphi(n)$:

"Der von n aufgespannte Teilbaum B_n hat nur endlich viele Knoten".

Da dies dann auch für die Wurzel w des Baums gilt und da $B = B_w$ ist, folgt die Behauptung.

Die Aussage $\forall n \in B.\ \varphi(n)$ wird durch noethersche Induktion über die angegebene Relation $\succ$ bewiesen. Wir zeigen also für alle $m \in B$:

"Wenn $\varphi(k)$ für alle $k \in M$ mit $m \succ k$ gilt, so gilt auch $\varphi(m)$."

Anders ausgedrückt, bedeutet dies:

"Wenn für alle direkten Kinder k von m der Baum B_k nur endlich viele Knoten hat, dann hat auch B_m nur endlich viele Knoten."

Man erhält $|B_m| = 1 + \sum_{k \text{ ist direktes Kind von } m} |B_k|$. Die Endlichkeit von $|B_k|$ für alle $k \in K$ folgt aus der Induktionshypothese. Da darüber hinaus der Verzweigungsgrad des Baums endlich ist, hat m nur endlich viele direkte Kinder k. Somit ist also auch $|B_m|$ endlich. □

4.2 Entscheidbarkeitsresultate zur Terminierung

Bevor wir Techniken entwickeln, um die Terminierung von TESen automatisch nachzuweisen, wollen wir uns zunächst über die Grenzen solcher Verfahren klar werden. Das Halteproblem ist nämlich in allen Turing-vollständigen Programmiersprachen unentscheidbar. TESe sind eine solche Turing-vollständige Sprache, denn jedes berechenbare Programm

[1] Formal benötigt man an dieser Stelle das *Auswahlaxiom*, da man diese Konstruktion *unendlich* oft fortsetzen muss.

kann durch ein TES ausgedrückt werden. Dies bedeutet zum einen, dass TESe in der Tat einen mächtigen Berechnungsformalismus darstellen und dass Techniken zur Analyse und Untersuchung von TESen auch für andere Programmiersprachen verwendbar sein können. Andererseits bedeutet es aber auch, dass es kein automatisches Verfahren gibt, das die Terminierung von TESen immer entscheiden kann. Wir müssen uns also mit hinreichenden Verfahren zufrieden geben, die möglichst viele terminierende TESe erkennen. Aus dem Fehlschlag eines solchen Verfahrens kann man dann aber nicht auf die Nicht-Terminierung des TES schließen.

Satz 4.2.1 (Unentscheidbarkeit des Halteproblems für TESe) *Sei $\mathcal{R}$ ein TES über Σ und $\mathcal{V}$ und sei $t \in \mathcal{T}(\Sigma, \mathcal{V})$. Das Problem, ob t keine unendliche Reduktion mit $\to_\mathcal{R}$ hat (Halteproblem), ist unentscheidbar, aber semi-entscheidbar. Das Problem, ob $\mathcal{R}$ terminiert (das universelle Halteproblem, d.h., die Frage, ob* alle *Terme nur endliche Reduktionen haben), ist nicht einmal semi-entscheidbar.*

Beweisskizze. Für jede Turingmaschine *TM* kann man ein TES $\mathcal{R}(TM)$ angeben, das die Arbeitsweise von *TM* simuliert (siehe [HL78] bzw. [BN98, Abschnitt 5.1.1]). Damit folgen die Unentscheidbarkeitsaussagen des Satzes aus der Unentscheidbarkeit des Halteproblems für Turingmaschinen und aus der Tatsache, dass das universelle Halteproblem für Turingmaschinen nicht einmal semi-entscheidbar ist [Her71].

Die Semi-Entscheidbarkeit des Halteproblems lässt sich wie folgt zeigen: Wir konstruieren einen Suchbaum wie in Abschnitt 3.1. Die Wurzel des Baums wird mit t markiert und jeder Knoten mit der Markierung u hat als Kinder die Knoten mit den Markierungen v für alle Terme v mit $u \to_\mathcal{R} v$. Aufgrund der Variablenbedingung $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ für alle Regeln $l \to r$ eines TES lässt sich jeder Term nur zu endlich vielen Termen in einem Schritt reduzieren. Der Suchbaum hat also einen endlichen Verzweigungsgrad. Der Term t terminiert gdw. jeder Pfad in dem Baum endliche Länge hat. Aufgrund des Lemmas von König (Satz 4.1.3) bedeutet das also, dass t terminiert gdw. der Suchbaum nur endlich viele Knoten hat. Das Semi-Entscheidungsverfahren baut also zu t den Suchbaum auf. Dieses Verfahren terminiert mit Erfolg gdw. t nur endliche Reduktionen mit $\to_\mathcal{R}$ hat. □

Allerdings existiert ein Spezialfall, in dem die Terminierung entscheidbar ist, nämlich bei TESen ohne Variablen auf den rechten Seiten der Regeln.

Beispiel 4.2.2 Wir betrachten das folgende TES, das keine Variablen auf den rechten Seiten seiner Regeln hat.

$$\begin{aligned}
\mathsf{and}(\mathsf{true}, \mathsf{true}) &\to \mathsf{true} \\
\mathsf{and}(x, \mathsf{false}) &\to \mathsf{false} \\
\mathsf{and}(\mathsf{false}, x) &\to \mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true})) \\
\mathsf{not}(\mathsf{false}) &\to \mathsf{true} \\
\mathsf{not}(\mathsf{true}) &\to \mathsf{and}(\mathsf{false}, \mathsf{false})
\end{aligned}$$

Das TES terminiert nicht, da es z.B. die folgende unendliche Reduktion gibt:

$$\begin{aligned} \mathsf{and}(\mathsf{false},\mathsf{false}) &\to_{\mathcal{R}} \mathsf{and}(\mathsf{true},\mathsf{not}(\mathsf{true})) \\ &\to_{\mathcal{R}} \mathsf{and}(\mathsf{true},\mathsf{and}(\mathsf{false},\mathsf{false})) \\ &\to_{\mathcal{R}} \mathsf{and}(\mathsf{true},\mathsf{and}(\mathsf{true},\mathsf{not}(\mathsf{true}))) \\ &\to_{\mathcal{R}} \ldots \end{aligned}$$

Zur Entwicklung eines Verfahrens für die Terminierungsüberprüfung solcher Systeme ist das folgende Lemma hilfreich. Hierbei ist wichtig, dass wir nur endliche TESe $\mathcal{R}$ betrachten.

Lemma 4.2.3 (Terminierung von TESen ohne Variablen auf rechten Seiten) *Sei $\mathcal{R}$ ein TES, wobei $\mathcal{V}(r) = \varnothing$ für alle $l \to r \in \mathcal{R}$ gilt. Dann terminiert das TES $\mathcal{R}$ gdw. es keine Regel $l \to r$ in $\mathcal{R}$ gibt, so dass $r \to^+_{\mathcal{R}} t$ für einen Term t gilt, der r als Teilterm enthält.*

Beweis. Die Richtung "$\Rightarrow$" ist offensichtlich, denn falls $t|_\pi = r$ ist, so erhält man die unendliche Reduktion $r \to^+_{\mathcal{R}} t = t[r]_\pi \to^+_{\mathcal{R}} t[t]_\pi = t[t[r]_\pi]_\pi \to^+_{\mathcal{R}} \ldots$

Wir zeigen die Richtung "$\Leftarrow$" durch Induktion über die Anzahl der Regeln in $\mathcal{R}$. Hierzu setzen wir voraus, dass es keine Regel $l \to r$ in $\mathcal{R}$ gibt, so dass $r \to^+_{\mathcal{R}} t$ für einen Term t gilt, der r als Teilterm enthält. Falls $\mathcal{R}$ leer ist, so ist die zu zeigende Aussage trivial, denn $\mathcal{R}$ terminiert.

Sei nun $\mathcal{R} \neq \varnothing$. Wir nehmen an, $\mathcal{R}$ terminiere nicht. Dann existiert ein minimaler Term t mit unendlicher Reduktion $t = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \ldots$. Das bedeutet also, dass die Reduktion von t nicht terminiert, aber alle seine echten Teilterme haben nur endliche Reduktionen. (Denn hätte ein echter Teilterm ebenfalls unendliche Reduktionen, so würden wir diesen Teilterm anstelle von t nehmen, etc.) Aufgrund der Minimalität folgt, dass in der unendlichen Reduktion irgendwann einmal ein Auswertungsschritt an der obersten Position ϵ stattfindet. Es gibt also ein $i \in \mathbb{N}$ mit $t_i = l\sigma$ und $t_{i+1} = r\sigma$ für eine Regel $l \to r \in \mathcal{R}$. Da r keine Variablen besitzt, folgt also $t_{i+1} = r$, d.h., es existiert eine unendliche Reduktion $r \to_{\mathcal{R}} t_{i+2} \to_{\mathcal{R}} \ldots$, die mit r beginnt. Wir betrachten nun zwei Fälle:

(1) Die Regel $l \to r$ wird in der unendlichen Reduktion $r \to_{\mathcal{R}} t_{i+2} \to_{\mathcal{R}} \ldots$ nicht verwendet. In diesem Fall terminiert bereits das TES $\mathcal{R} \setminus \{l \to r\}$ nicht. Dies ist aber ein Widerspruch zur Induktionshypothese.

(2) Ansonsten wird die Regel $l \to r$ in der unendlichen Reduktion von r verwendet, d.h., es existiert ein $j \geq i+1$ mit $r \to^*_{\mathcal{R}} t_j = t_j[l\sigma']_\pi \to_{\mathcal{R}} t_j[r]_\pi = t_{j+1}$. Dies widerspricht der Voraussetzung. □

Beispielsweise hat das nicht-terminierende System aus Bsp. 4.2.2 die rechte Seite $\mathsf{and}(\mathsf{false},\mathsf{false})$, die zu dem Term $\mathsf{and}(\mathsf{true},\mathsf{and}(\mathsf{false},\mathsf{false}))$ reduziert werden kann, der diese rechte Seite als Teilterm enthält.

Nun können wir ein Verfahren angeben, das die Terminierung bei TESen ohne Variablen auf rechten Seiten entscheiden kann: Für ein TES $\mathcal{R} = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ mit $\mathcal{V}(r_i) = \varnothing$ für $1 \leq i \leq n$ generiert man erst alle Reduktionen der Länge 1, die mit $r_1, \ldots, r_n$ beginnen,

dann alle Reduktionen der Länge 2, etc. Die führt man solange durch, bis entweder keine weiteren Reduktionen mehr möglich sind (in diesem Fall terminiert das TES) oder bis man einen Term t mit $r_i \to_{\mathcal{R}}^+ t$ gefunden hat, der r_i als Teilterm enthält (in diesem Fall terminiert das TES nicht). Im Unterschied zu einem entsprechenden Verfahren für die Terminierung allgemeiner TESe (ähnlich zu dem Suchbaumverfahren aus dem Beweis von Satz 4.2.1) hat man hier also zwei Einschränkungen:

1. Man muss nicht für alle Terme t einen Suchbaum mit den durch Reduktionen erhältlichen Termen konstruieren, sondern nur für die (endlich vielen) rechten Seiten der Regeln des TES.

2. Man kann die Konstruktion der (potentiell unendlichen) Suchbäume abbrechen, sobald man im Suchbaum von r_i einen Term t findet, der r_i als Teilterm enthält. Es ist aufgrund von Lemma 4.2.3 sicher gestellt, dass dies stets für ein r_i der Fall ist, wenn $\mathcal{R}$ nicht terminiert.

Algorithmus RIGHT-GROUND-TERMINIERUNG($\mathcal{R}$)
Eingabe: Ein TES $\mathcal{R}$ ohne Variablen auf rechten Seiten.
Ausgabe: "*True*", falls $\mathcal{R}$ terminiert und sonst "*False*".

1. Für $\mathcal{R} = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ sei $T_i = \{r_i\}$, $1 \leq i \leq n$.
2. Für alle i setze $T_i = \{t \mid s \in T_i, s \to_{\mathcal{R}} t\}$.
3. Falls $T_i = \varnothing$ für alle i, dann gib "*True*" aus und breche ab.
4. Falls es ein i und ein $t \in T_i$ gibt, so dass $t \trianglerighteq r_i$, dann gib "*False*" aus und breche ab.
5. Gehe zu Schritt 2.

Beispiel 4.2.4 Wir betrachten wieder das TES aus Bsp. 4.2.2. In Schritt 1 setzen wir $T_1 = \{\mathsf{true}\}, T_2 = \{\mathsf{false}\}, T_3 = \{\mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true}))\}, T_4 = \{\mathsf{true}\}, T_5 = \{\mathsf{and}(\mathsf{false}, \mathsf{false})\}$.

In Schritt 2 ersetzt man die Terme in diesen Mengen durch diejenigen Terme, die man durch einen Ersetzungsschritt erhält. So ergibt sich $T_1 = T_2 = T_4 = \varnothing$, $T_3 = \{\mathsf{and}(\mathsf{true}, \mathsf{and}(\mathsf{false}, \mathsf{false}))\}$, $T_5 = \{\mathsf{false}, \mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true}))\}$.

In Schritt 4 stellt man fest, dass $\mathsf{and}(\mathsf{true}, \mathsf{and}(\mathsf{false}, \mathsf{false})) \not\trianglerighteq \mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true}))$ und dass sowohl $\mathsf{false} \not\trianglerighteq \mathsf{and}(\mathsf{false}, \mathsf{false})$ als auch $\mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true})) \not\trianglerighteq \mathsf{and}(\mathsf{false}, \mathsf{false})$ gilt.

Somit erreicht man Schritt 5 und springt zurück zu Schritt 2. Im Folgenden bleiben T_1, T_2 und T_4 leer und man erhält $T_3 = \{\mathsf{and}(\mathsf{true}, \mathsf{false}), \mathsf{and}(\mathsf{true}, \mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true})))\}$ und $T_5 = \{\mathsf{and}(\mathsf{true}, \mathsf{and}(\mathsf{false}, \mathsf{false}))\}$.

In Schritt 4 bemerkt man nun, dass der Term $\mathsf{and}(\mathsf{true}, \mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true})))$ aus T_3 die ursprüngliche rechte Seite $\mathsf{and}(\mathsf{true}, \mathsf{not}(\mathsf{true}))$ als Teilterm enthält. Ebenso enthält der Term $\mathsf{and}(\mathsf{true}, \mathsf{and}(\mathsf{false}, \mathsf{false}))$ aus T_5 die zugehörige rechte Seite $\mathsf{and}(\mathsf{false}, \mathsf{false})$. Somit gibt der Algorithmus "*False*" aus und terminiert.

Satz 4.2.5 (Entscheidbarkeit der Terminierung) *Sei $\mathcal{R}$ ein TES, wobei $\mathcal{V}(r) = \varnothing$ für alle $l \to r \in \mathcal{R}$. Dann ist entscheidbar, ob $\mathcal{R}$ terminiert und der Algorithmus* RIGHT-GROUND-TERMINIERUNG *ist ein Entscheidungsverfahren.*

Beweis. Wir betrachten zwei Fälle. Falls $\mathcal{R}$ terminiert, so gibt es nach Lemma 4.2.3 keinen Term t mit $r_i \to_{\mathcal{R}}^+ t$ und $t \trianglerighteq r_i$. Somit gibt der Algorithmus RIGHT-GROUND-TERMINIERUNG also nicht "*False*" aus. Außerdem hat dann kein Term r_i unendlich lange Reduktionen. Da außerdem der Verzweigungsgrad der Ersetzungsrelation $\to_{\mathcal{R}}$ endlich ist, folgt aufgrund des Lemmas von König (Satz 4.1.3), dass die Suchbäume für alle r_i endlich sind. Mit anderen Worten, nach endlich vielen Schritten sind alle $T_i = \varnothing$ und der Algorithmus terminiert mit der Ausgabe "*True*".

Falls $\mathcal{R}$ nicht terminiert, so existiert nach Lemma 4.2.3 ein $i \in \{1, \ldots, n\}$ mit $r_i \to_{\mathcal{R}}^+ t$ und $t \trianglerighteq r_i$. Nach endlich vielen Schritten gilt also $t \in T_i$ und der Algorithmus terminiert mit der Ausgabe "*False*". □

4.3 Terminierung mit Reduktionsrelationen

Nachdem wir ein Entscheidungsverfahren für die Terminierung bei TESen ohne Variablen auf rechten Seiten kennen gelernt haben, wollen wir nun wieder den Fall allgemeiner TESe untersuchen. Aufgrund der Unentscheidbarkeit des Halteproblems (Satz 4.2.1) wird jedes automatische Verfahren unvollständig sein. Unser Ziel ist aber natürlich, ein Verfahren zu entwickeln, das zumindest bei vielen praktisch interessanten TESen (und Programmen) die Terminierung nachweisen kann.

Eine Grundidee, um die Terminierung eines TES $\mathcal{R}$ (bzw. die Fundiertheit der Ersetzungsrelation $\to_{\mathcal{R}}$) nachzuweisen, ist der Ansatz, eine fundierte Relation $\succ$ zu finden, in der die Ersetzungsrelation enthalten ist. Mit anderen Worten, wir suchen nach einer Relation $\succ$, so dass für alle Terme s, t aus $s \to_{\mathcal{R}} t$ jeweils $s \succ t$ folgt. Offensichtlich ist dies für die Terminierung hinreichend, d.h., aus der Fundiertheit von $\succ$ folgt dann die Fundiertheit von $\to_{\mathcal{R}}$.

Das Ziel ist hierbei, eine solche Relation $\succ$ *automatisch* zu finden. Allerdings gibt es natürlich im allgemeinen unendlich viele Terme s und t, so dass $s \to_{\mathcal{R}} t$ gilt. Es ist daher automatisch nicht möglich, für all diese Terme zu untersuchen, ob dann auch wirklich $s \succ t$ folgt.

Wir müssen daher versuchen, diese Eigenschaft schon durch Betrachtung von nur endlich vielen Termpaaren s, t sicherzustellen. Die Ersetzungsrelation $\to_{\mathcal{R}}$ ist ja durch die (endlich vielen) Regeln festgelegt, denn es gilt $s \to_{\mathcal{R}} t$ genau dann, wenn $s|_\pi = l\sigma$ und $t = s[r\sigma]_\pi$ für eine Regel $l \to r \in \mathcal{R}$. Wir verlangen daher

$$l \succ r \quad \text{für alle Regeln } l \to r \in \mathcal{R}. \tag{4.7}$$

Diese Forderung alleine stellt natürlich noch nicht sicher, dass auch wirklich aus $s \to_{\mathcal{R}} t$ folgt, dass dann auch $s \succ t$ gilt.

Als Beispiel betrachten wir das TES mit der Regel

$$\mathsf{endless}(x) \;\to\; \mathsf{endless}(\mathsf{succ}(x)).$$

Eine fundierte Relation, die die Bedingung (4.7) erfüllt, lässt sich bei diesem TES leicht angeben. Die Bedingung $\mathsf{endless}(x) \succ \mathsf{endless}(\mathsf{succ}(x))$ wird z.B. von der Relation erfüllt,

bei der $s \succ t$ genau dann gilt, wenn $s = \mathsf{endless}(x)$ und $t = \mathsf{endless}(\mathsf{succ}(x))$ ist. Dennoch terminiert der Algorithmus endless natürlich nicht.

Der Grund für dieses Problem ist, dass bei der Auswertung ja eine *Instanz* $l\sigma$ der linken Seite l durch die entsprechende Instanz $r\sigma$ der rechten Seite ersetzt wird. Man muss daher zusätzlich sicherstellen, dass aus $l \succ r$ jeweils auch $l\sigma \succ r\sigma$ folgt. Die Relation $\succ$ muss also *stabil*, d.h., abgeschlossen unter Substitutionen, sein.

Wenn man auch die Stabilität von $\succ$ fordert, dann lässt sich die Terminierung des endless-TES nicht mehr fälschlicherweise "beweisen". Mit anderen Worten, es gibt keine fundierte und stabile Relation $\succ$, die $\mathsf{endless}(x) \succ \mathsf{endless}(\mathsf{succ}(x))$ erfüllt. Der Grund ist, dass für jede stabile Relation mit $\mathsf{endless}(x) \succ \mathsf{endless}(\mathsf{succ}(x))$ auch $\mathsf{endless}(x) \succ \mathsf{endless}(\mathsf{succ}(x)) \succ \mathsf{endless}(\mathsf{succ}(\mathsf{succ}(x))) \succ \ldots$ gelten würde, was der Fundiertheit widerspricht.

Ein Beispiel für eine fundierte Relation, die auch stabil ist, ist die echte Teiltermrelation $\rhd$. Hier gilt $s \rhd t$ gdw. t ein echter Teilterm von s ist. Dann folgt aber auch $s\sigma \rhd t\sigma$ für alle Substitutionen σ, da dann jeweils $t\sigma$ auch ein echter Teilterm von $s\sigma$ ist. Die fundierte Relation $\to_{||}$, die Terme nach der Anzahl der darin vorkommenden Symbole vergleicht, ist hingegen nicht stabil. Es gilt beispielsweise $\mathsf{succ}(x) \to_{||} y$, aber $\mathsf{succ}(x)\sigma \not\to_{||} y\sigma$ für die Substitution $\sigma = \{y/\mathsf{succ}(\mathsf{succ}(x))\}$.

Wir benötigen allerdings noch eine weitere Forderung neben der Stabilität, um sicherzustellen, dass aus (4.7) tatsächlich $s \succ t$ für alle Terme mit $s \to_{\mathcal{R}} t$ folgt. Als Beispiel betrachten wir das TES mit der folgenden Regel.

$$\mathsf{infty}(x) \;\to\; \mathsf{succ}(\mathsf{infty}(x))$$

Es existiert eine fundierte und stabile Relation $\succ$, die die Bedingung (4.7) für dieses TES erfüllt, d.h., $\mathsf{infty}(x) \succ \mathsf{succ}(\mathsf{infty}(x))$. Ein Beispiel ist die Relation $\succ$, die zwei Terme nur nach ihrem äußersten Funktionssymbol vergleicht. Wir definieren also $s \succ t$ gdw. $s = \mathsf{infty}(\ldots)$ und $t = \mathsf{succ}(\ldots)$. Die Fundiertheit und Stabilität der Relation ist offensichtlich. Dennoch terminiert der Algorithmus infty nicht.

Der Grund ist, dass bei der Ersetzungsrelation die Ersetzung von instantiierten linken Seiten durch instantiierte rechte Seiten ja für beliebige *Teilterme* möglich ist. Die Stabilität von $\succ$ garantiert $l\sigma \succ r\sigma$ für alle Substitutionen σ; wir benötigen aber darüber hinaus noch $q[l\sigma]_\pi \succ q[r\sigma]_\pi$ für alle Stellen π in allen Termen q. Eine Ungleichung wie $l\sigma \succ r\sigma$ muss also auch dann noch gelten, wenn $l\sigma$ und $r\sigma$ in einem beliebigen Kontext auftreten. Die Relation $\succ$ muss also auch noch *monoton*, d.h., abgeschlossen unter Kontexten, sein.

Wenn wir die Monotonie fordern, kann die Terminierung des infty-TES nicht mehr fälschlicherweise nachgewiesen werden. Es existiert also keine monotone und fundierte Relation $\succ$, die $\mathsf{infty}(x) \succ \mathsf{succ}(\mathsf{infty}(x))$ erfüllt. Aus der Monotonie folgt nämlich $\mathsf{infty}(x) \succ \mathsf{succ}(\mathsf{infty}(x)) \succ \mathsf{succ}(\mathsf{succ}(\mathsf{infty}(x))) \succ \ldots$, was der Fundiertheit widerspricht.

Die echte Teiltermrelation ist zwar stabil, aber nicht monoton. Der Grund ist, dass zwar x ein echter Teilterm von $\mathsf{succ}(x)$ ist, d.h., $\mathsf{succ}(x) \rhd x$, aber es gilt nicht $\mathsf{infty}(\mathsf{succ}(x)) \rhd \mathsf{infty}(x)$, denn $\mathsf{infty}(x)$ ist kein Teilterm von $\mathsf{infty}(\mathsf{succ}(x))$. Die Relation $\to_{||}$ ist hingegen zwar nicht stabil, aber dafür monoton. Wenn nämlich t weniger Symbole als s enthält (d.h. $s \to_{||} t$), so enthält auch $q[t]_\pi$ weniger Symbole als $q[s]_\pi$.

Ein Beispiel für eine Relation, die sowohl fundiert und stabil als auch monoton ist, ist die folgende sogenannte *Einbettungsordnung*.

Definition 4.3.1 (Einbettungsordnung) *Für eine Signatur Σ ist die Einbettungsordnung $\succ_{emb}$ über $\mathcal{T}(\Sigma, \mathcal{V})$ definiert als $s \succ_{emb} t$ gdw.*

- *$s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{emb} t$ für ein $i \in \{1, \ldots, n\}$ oder*
- *$s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, $s_i \succ_{emb} t_i$ für ein $i \in \{1, \ldots, n\}$ und $s_j \succeq_{emb} t_j$ für alle $j \in \{1, \ldots, n\}$ mit $j \neq i$.*

Hierbei bezeichnet $\succeq_{emb}$ die reflexive Hülle von $\succ_{emb}$, d.h., die Relation mit $s \succeq_{emb} t$ gdw. $s \succ_{emb} t$ oder $s = t$.

Beispielsweise erhalten wir also

$$\mathsf{succ}(\mathsf{plus}(\mathsf{infty}(\mathsf{succ}(x)), y)) \succ_{emb} \mathsf{plus}(\mathsf{infty}(x), y),$$

denn es gilt $\mathsf{plus}(\mathsf{infty}(\mathsf{succ}(x)), y) \succ_{emb} \mathsf{plus}(\mathsf{infty}(x), y)$. Der Grund ist, dass $y \succeq_{emb} y$ und $\mathsf{infty}(\mathsf{succ}(x)) \succ_{emb} \mathsf{infty}(x)$ gelten (denn wir haben $\mathsf{succ}(x) \succ_{emb} x$ wegen $x \succeq_{emb} x$).

Lemma 4.3.2 (Eigenschaften der Einbettungsordnung) *Die Einbettungsordnung $\succ_{emb}$ ist fundiert, stabil, monoton und transitiv.*

Beweis. Die Fundiertheit der Einbettungsordnung ist offensichtlich, da $s \succ_{emb} t$ impliziert, dass t weniger Symbole als s enthält. (Dies lässt sich durch eine leichte strukturelle Induktion über s beweisen.) Somit ist $\succ_{emb}$ eine Teilrelation von $\rightarrow_{||}$ und damit fundiert.

Die Stabilität zeigt man durch strukturelle Induktion über s. Falls $s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{emb} t$ gilt, so gilt auch $s_i\sigma \succeq_{emb} t\sigma$ nach der Induktionshypothese und somit $s\sigma \succ_{emb} t\sigma$. Falls $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, $s_i \succ_{emb} t_i$ und $s_j \succeq_{emb} t_j$ für $j \neq i$, so folgt aus der Induktionshypothese $s_i\sigma \succ_{emb} t_i\sigma$ und $s_j\sigma \succeq_{emb} t_j\sigma$ für $j \neq i$. Damit gilt auch $s\sigma = f(s_1\sigma, \ldots, s_n\sigma) \succ_{emb} f(t_1\sigma, \ldots, t_n\sigma) = t\sigma$.

Die Monotonie (d.h., dass aus $s \succ_{emb} t$ auch $q[s]_\pi \succ_{emb} q[t]_\pi$ folgt), beweist man durch Induktion über den Term q (bzw. über die Stelle π). Im Fall $\pi = \epsilon$ ist die Aussage trivial. Ansonsten hat π die Gestalt $i\,\pi'$ und q ist $f(q_1, \ldots, q_n)$. Damit gilt $q[s]_\pi = f(q_1, \ldots, q_{i-1}, q_i[s]_{\pi'}, q_{i+1}, \ldots, q_n) \succ_{emb} f(q_1, \ldots, q_{i-1}, q_i[t]_{\pi'}, q_{i+1}, \ldots, q_n) = q[t]_\pi$ nach der Induktionshypothese $q_i[s]_{\pi'} \succ_{emb} q_i[t]_{\pi'}$.

Schließlich beweisen wir auch die Transitivität von $\succ_{emb}$. Hierzu zeigen wir durch strukturelle Induktion über den Term s, dass aus $s \succ_{emb} t \succ_{emb} r$ folgt, dass auch $s \succ_{emb} r$ gilt:

Falls $s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{emb} t$ für ein i gilt, so folgt damit $s_i \succeq_{emb} t \succ_{emb} r$ und daher nach der Induktionshypothese $s_i \succ_{emb} r$. Dies ergibt wiederum $s = f(s_1, \ldots, s_n) \succ_{emb} r$ nach der Definition der Einbettungsordnung.

Ansonsten gilt $s = f(s_1, \ldots, s_i, \ldots, s_n)$, $t = f(t_1, \ldots, t_i, \ldots, t_n)$, $s_i \succ_{emb} t_i$ und $s_j \succeq_{emb} t_j$ für alle $j \neq i$. Falls es ein $k \in \{1, \ldots, n\}$ mit $t_k \succeq_{emb} r$ gibt, so folgt nach der Induktionshypothese auch $s_k \succeq_{emb} r$ und damit $s = f(s_1, \ldots, s_n) \succ_{emb} r$ nach der Definition der Einbettungsordnung. Ansonsten gilt $r = f(r_1, \ldots, r_n)$ und $t_k \succeq_{emb} r_k$ für alle $k \in \{1, \ldots, n\}$. Die Induktionshypothese ergibt $s_i \succ_{emb} r_i$ und $s_j \succeq_{emb} r_j$ für alle $j \neq i$. Daraus folgt $s = f(s_1, \ldots, s_n) \succ_{emb} f(r_1, \ldots, r_n) = r$. □

Relationen, die fundiert, monoton und stabil sind, bezeichnet man als *Reduktionsrelationen*. Ist solch eine Relation wie $\succ_{emb}$ darüber hinaus auch noch transitiv, spricht man von

Reduktionsordnungen. (Wie üblich ist eine *Ordnung* eine transitive und antisymmetrische[2] Relation. Aus der Fundiertheit folgt natürlich sofort die Asymmetrie[3] und damit auch die Antisymmetrie.)

Definition 4.3.3 (Reduktionsrelation und -ordnung) *Eine Relation $\succ$ über Termen, die fundiert, stabil und monoton ist, heißt Reduktionsrelation. Eine transitive Reduktionsrelation heißt Reduktionsordnung.*

Mit Hilfe von Reduktionsrelationen lässt sich nun tatsächlich die Terminierung von TESen (und damit auch von Programmen) nachweisen. Man muss lediglich untersuchen, ob für alle definierenden Gleichungen die linke Seite jeweils größer als die rechte ist, vgl. (4.7). Es handelt sich hierbei nicht nur um ein hinreichendes, sondern sogar um ein notwendiges Kriterium für die Terminierung.

Satz 4.3.4 (Terminierungskriterium mit Reduktionsrelationen [MN70]) *Ein TES $\mathcal{R}$ terminiert genau dann, wenn es eine Reduktionsrelation $\succ$ gibt, so dass $l \succ r$ für alle Regeln $l \to r \in \mathcal{R}$ gilt.*

Beweis.

"$\Leftarrow$": Sei $\succ$ eine Reduktionsrelation, so dass $l \succ r$ für alle Regeln $l \to r \in \mathcal{R}$ gilt. Dann folgt für alle Terme mit $s \to_{\mathcal{R}} t$ auch $s \succ t$. Der Grund ist, dass $s \to_{\mathcal{R}} t$ bedeutet, dass es eine Stelle π in s, eine Substitution σ und eine Regel $l \to r$ gibt mit $s|_\pi = l\sigma$ und $t = s[r\sigma]_\pi$. Aufgrund der Stabilität von $\succ$ gilt $l\sigma \succ r\sigma$ und aufgrund der Monotonie gilt auch $s = s[l\sigma]_\pi \succ s[r\sigma]_\pi = t$.

Da $\to_{\mathcal{R}}$ demnach eine Teilrelation von $\succ$ ist, folgt aus der Fundiertheit von $\succ$ daher auch die Fundiertheit von $\to_{\mathcal{R}}$.

"$\Rightarrow$": Sei $\succ$ die Relation $\to_{\mathcal{R}}$. Dann gilt offensichtlich $l \succ r$ für alle Regeln $l \to r \in \mathcal{R}$. Die Stabilität und Monotonie von $\succ$ folgt aus Lemma 3.1.13 und die Fundiertheit von $\succ$ folgt aus der Terminierung von $\mathcal{R}$. □

Sofern man eine zugrunde liegende Reduktionsrelation $\succ$ verwendet, bei der für alle Terme s, t entscheidbar ist, ob $s \succ t$ gilt, ergibt sich mit Hilfe von Satz 4.3.4 sofort ein Verfahren für automatische Terminierungsbeweise. Das Terminierungsverfahren untersucht einfach, ob $l \succ r$ für alle Regeln $l \to r$ des TES gilt. In diesem Fall ist die Terminierung des TES bewiesen. Obwohl Satz 4.3.4 ja ein hinreichendes und notwendiges Terminierungskriterium darstellt, ist dies aber dennoch ein unvollständiges Beweisverfahren. Gibt es also eine Regel mit $l \not\succ r$, so kann unser Verfahren keine Aussage über die Terminierung des TES machen. Der Grund ist, dass es ja eine andere Reduktionsrelation $\succ'$ geben könnte, bei der tatsächlich $l \succ' r$ für alle Regeln gilt.

Ein Algorithmus zur Entscheidung von $s \succ_{emb} t$ für alle Terme s und t lässt sich leicht angeben, da die beiden Bedingungen von Def. 4.3.1 wie in einem rekursiven Programm

[2]Eine Relation $\to$ heißt *antisymmetrisch* gdw. aus $s \to t$ und $t \to s$ folgt, dass $s = t$ gilt.

[3]Eine Relation $\to$ heißt *asymmetrisch* gdw. es keine Objekte s und t mit $s \to t$ und $t \to s$ gibt.

abgearbeitet werden können. Die Terminierung dieses rekursiven Programms folgt daraus, dass die rekursiven Argumente (s_i und t bei der ersten Bedingung und s_i und t_i bzw. s_j und t_j bei der zweiten Bedingung) immer kleinere Terme werden. Die Einbettungsordnung lässt sich daher als zugrunde liegende Reduktionsrelation in einem Terminierungsverfahren wie oben skizziert verwenden. Dieses Beweisverfahren ist bereits in der Lage, die Terminierung einfacher Programme nachzuweisen.

Beispiel 4.3.5 Betrachten wir ein TES zur Subtraktion zweier natürlicher Zahlen.

$$\begin{aligned} \mathsf{minus}(x, \mathcal{O}) &\to x \\ \mathsf{minus}(\mathcal{O}, \mathsf{succ}(y)) &\to \mathcal{O} \\ \mathsf{minus}(\mathsf{succ}(x), \mathsf{succ}(y)) &\to \mathsf{minus}(x, y) \end{aligned}$$

Wenn wir die Einbettungsordnung $\succ_{emb}$ als zugrunde liegende Reduktionsrelation verwenden, so müssen wir nun nachweisen, dass bei jeder Regel die linke Seite größer als die rechte ist, d.h.

$$\begin{aligned} \mathsf{minus}(x, \mathcal{O}) &\succ_{emb} x \\ \mathsf{minus}(\mathcal{O}, \mathsf{succ}(y)) &\succ_{emb} \mathcal{O} \\ \mathsf{minus}(\mathsf{succ}(x), \mathsf{succ}(y)) &\succ_{emb} \mathsf{minus}(x, y) \end{aligned}$$

Dies lässt sich sofort (automatisch) zeigen, so dass die Terminierung des TES bewiesen ist.

Allerdings ist unser Terminierungsverfahren noch recht schwach. Die Terminierung des plus-TES lässt sich beispielsweise mit der Einbettungsordnung nicht zeigen. Wie üblich ist plus durch die folgenden Regeln definiert.

$$\begin{aligned} \mathsf{plus}(\mathcal{O}, y) &\to y \\ \mathsf{plus}(\mathsf{succ}(x), y) &\to \mathsf{succ}(\mathsf{plus}(x, y)) \end{aligned}$$

Man müsste also die folgenden Ungleichungen beweisen:

$$\begin{aligned} \mathsf{plus}(\mathcal{O}, y) &\succ_{emb} y \\ \mathsf{plus}(\mathsf{succ}(x), y) &\succ_{emb} \mathsf{succ}(\mathsf{plus}(x, y)) \end{aligned}$$

Es gilt aber $\mathsf{plus}(\mathsf{succ}(x), y) \not\succ_{emb} \mathsf{succ}(\mathsf{plus}(x, y))$, denn $\mathsf{succ}(\mathsf{plus}(x, y))$ ist weder in $\mathsf{succ}(x)$ noch in y eingebettet und die beiden Terme haben auch nicht das gleiche äußere Funktionssymbol.

Das Beispiel plus zeigt, dass die Verwendung der Einbettungsordnung eine zu große Einschränkung ist, um ein praktisch einsetzbares Terminierungsverfahren zu erhalten. Im nächsten Abschnitt werden wir daher weitere Reduktionsrelationen einführen, die ebenfalls entscheidbar sind und mit denen das Terminierungsverfahren wesentlich leistungsfähiger wird.

4.4 Simplifikationsordnungen und rekursive Pfadordnungen

Die Einbettungsordnung ist ein erster Schritt zur Entwicklung von Reduktionsordnungen, die zum Terminierungsnachweis geeignet sind. Wie das Beispiel plus zeigt, muss man diese Ordnung jedoch noch erweitern, um praktische Beispiele behandeln zu können. Solche Reduktionsordnungen, die die Einbettungsordnung enthalten, bezeichnet man als *Simplifikationsordnungen.*

Definition 4.4.1 (Simplifikationsordnung [Der87]) *Eine Reduktionsordnung $\succ$ mit $s \succ t$ für alle $s \succ_{emb} t$ heißt* Simplifikationsordnung.

Man kann zeigen, dass man in dieser Definition die Fundiertheit gar nicht fordern muss, sondern sie folgt bereits daraus, dass Simplifikationsordnungen die Einbettungsordnung enthalten. Der genaue Zusammenhang wird im folgenden Satz beschrieben. Hierbei ist Teil (a) der bekannte Satz von Kruskal [Kru60]. Hierbei ist entscheidend, dass die Signatur Σ (d.h., die Menge der Funktionssymbole) endlich ist.

Satz 4.4.2 (Satz von Kruskal)

(a) Für jede unendliche Folge von Grundtermen $t_0, t_1, t_2, \ldots$ existieren $i, j \in \mathbb{N}$ mit $i < j$ und $t_i \preceq_{emb} t_j$.

(b) Jede stabile, monotone, transitive Relation $\succ$, die die Teiltermeigenschaft *$f(x_1, \ldots, x_n) \succ x_i$ für alle $1 \leq i \leq n$ erfüllt, enthält die Einbettungsordnung (d.h., es gilt $\succ_{emb} \subseteq \succ$).*

(c) Jede stabile, monotone, transitive, **irreflexive** *Relation $\succ$, die die Teiltermeigenschaft $f(x_1, \ldots, x_n) \succ x_i$ für alle $1 \leq i \leq n$ erfüllt, ist fundiert (und damit eine Simplifikationsordnung).*

Beweis.

(a) Für den Beweis des Satzes von Kruskal (Teil (a)) wird auf [BN98, Abschnitt 5.4] verwiesen.

(b) Sei $s \succ_{emb} t$. Wir zeigen $s \succ t$ durch strukturelle Induktion über s.

Falls $s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{emb} t$ für ein $i \in \{1, \ldots, n\}$ gilt, so folgt $s_i \succeq t$ aufgrund der Induktionshypothese. Aus der Teiltermeigenschaft und der Stabilität von $\succ$ folgt $s = f(s_1, \ldots, s_n) \succ s_i$ und somit ergibt sich $s \succ t$ aufgrund der Transitivität von $\succ$.

Sei nun $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, $s_i \succ_{emb} t_i$ für ein $i \in \{1, \ldots, n\}$ und $s_j \succeq_{emb} t_j$ für alle $j \in \{1, \ldots, n\}$ mit $j \neq i$. Aus der Induktionshypothese folgt somit $s_i \succ t_i$ und $s_j \succeq t_j$ für alle $j \neq i$. Aufgrund der Monotonie und Transitivität von $\succ$ erhält man daher $s = f(s_1, \ldots, s_n) \succ f(t_1, \ldots, t_n) = t$.

(c) Wir nehmen an, die Relation $\succ$ sei nicht fundiert. Dann existiert eine unendliche absteigende Folge von Termen

$$t_0 \succ t_1 \succ t_2 \succ \ldots$$

Sei σ eine Substitution, die alle Variablen der t_i durch Grundterme ersetzt (wir verlangen stets $\Sigma_0 \neq \varnothing$, d.h., dass Grundterme existieren). Wegen der Stabilität von $\succ$ gilt dann

$$t_0\sigma \succ t_1\sigma \succ t_2\sigma \succ \ldots,$$

wobei alle $t_i\sigma$ Grundterme sind. Aufgrund des Satzes von Kruskal (Teil (a)) gibt es also $i < j$ mit $t_i\sigma \preceq_{emb} t_j\sigma$. Da $\preceq_{emb} \subseteq \preceq$ ist (Teil (b)), hat man auch $t_i\sigma \preceq t_j\sigma$. Damit folgt aufgrund der Transitivität $t_i\sigma \succ t_j\sigma \succeq t_i\sigma$, d.h., $t_i\sigma \succ t_i\sigma$ im Widerspruch zur Irreflexivität von $\succ$. □

Im Folgenden werden wir Reduktionsordnungen $\succ$ definieren, die mächtiger als die Einbettungsordnung sind, und bei denen man dennoch weiterhin automatisch überprüfen kann, ob $s \succ t$ für zwei Terme gilt. Um zu beweisen, dass es sich bei einer solchen Relation tatsächlich um eine Reduktionsordnung handelt, genügt es dann jeweils, zu zeigen, dass $\succ$ stabil, monoton, transitiv und irreflexiv ist und darüber hinaus die Teiltermeigenschaft $f(x_1, \ldots, x_n) \succ x_i$ erfüllt. Nach dem Satz von Kruskal (bzw. nach Satz 4.4.2 (c)) folgt daraus dann, dass $\succ$ eine Simplifikationsordnung und somit auch eine Reduktionsordnung ist.

Zur Definition einer Ordnung, die die Einbettungsordnung erweitert, müssen wir insbesondere angeben, wie man zwei Terme $f(s_1, \ldots, s_n)$ und $f(t_1, \ldots, t_n)$ mit dem gleichen Funktionssymbol vergleichen soll. In der Einbettungsordnung wurde hier $s_i \succ t_i$ für ein i und $s_j \succeq t_j$ für alle $j \neq i$ gefordert. Diese Forderung lässt sich aber abschwächen, indem man stattdessen die Argumente der beiden f-Terme entweder *lexikographisch* oder als *Multimenge* vergleicht. Die erste Form des Vergleichs führt zur sogenannten *lexikographischen Pfadordnung* und die zweite Form führt zur *rekursiven Pfadordnung*.

Hierzu werden wir nun zwei Konstruktionstechniken einführen, mit denen man aus einfachen Ordnungen komplexere Ordnungen konstruieren kann. Die lexikographische Kombination dient dazu, zwei Relationen auf den Mengen T_1 und T_2 zu einer Relation auf Paaren aus $T_1 \times T_2$ zu kombinieren.

Definition 4.4.3 (Lexikographische Kombination von Relationen) *Sei $\succ_1$ eine Relation auf der Menge T_1 und $\succ_2$ eine Relation auf der Menge T_2 (d.h., $\succ_i \subseteq T_i \times T_i$). Dann wird die lexikographische Kombination $\succ_{1\times 2}$ auf $T_1 \times T_2$ wie folgt definiert:*

$$(s_1, s_2) \succ_{1\times 2} (t_1, t_2) \quad \textit{gdw.} \quad s_1 \succ_1 t_1 \quad \textit{oder} \quad (s_1 = t_1 \textit{ und } s_2 \succ_2 t_2).$$

Analog dazu kann man die lexikographische Kombination von mehreren Relationen $\succ_1$, $\succ_2, \ldots, \succ_n$ als die Kombination von $\succ_1$ mit der lexikographischen Kombination von $\succ_2$, $\ldots, \succ_n$ definieren, d.h.

$$(s_1, \ldots, s_n) \succ_{1\times\ldots\times n} (t_1, \ldots, t_n) \quad \textit{gdw.} \quad \textit{es existiert } i \in \{1, \ldots, n\} \textit{ mit } s_i \succ_i t_i \textit{ und } s_j = t_j \textit{ für } 1 \leq j < i.$$

Die n-fache lexikographische Kombination einer Relation $\succ$ mit sich selbst wird als $\succ_{lex}^n$ bezeichnet (d.h., dies entspricht dem obigen Fall, wenn $\succ_1 = \ldots = \succ_n$ ist).

Solch eine Form des Vergleichs heißt deswegen *lexikographisch*, weil sie (in etwa) der Relation entspricht, nach der Einträge in einem Lexikon geordnet sind. (Ein Begriff "$c_1 \ldots c_n$" steht vor einem anderen Begriff "$d_1 \ldots d_m$" im Lexikon, wenn der i-te Buchstabe c_i des ersten Worts vor dem i-ten Buchstaben d_i des zweiten Worts im Alphabet kommt und die Buchstaben der beiden Wörter vor dem i-ten Buchstaben gleich sind (d.h., $c_j = d_j$ für alle $1 \leq j < i$). Die Buchstaben hinter dem i-ten Buchstaben können aber in den beiden Wörtern beliebig sein.

Beispiel 4.4.4 Sei $>_{\mathbb{N}}$ wieder die übliche "größer"-Relation auf natürlichen Zahlen und sei $>_{alph}$ die alphabetische Relation auf Buchstaben. Dann gilt $(3, c) \succ (2, b)$, $(3, c) \succ (3, d)$, aber $(3, c) \not\succ (4, d)$ für die lexikographische Kombination $\succ$ der beiden Relationen $>_{\mathbb{N}}$ und $>_{alph}$. Außerdem gilt $hans\ (\succ_{alph})^4_{lex}\ hugo\ (\succ_{alph})^4_{lex}\ kurt$.

Das folgende grundlegende Resultat zeigt, dass bei der lexikographischen Kombination von fundierten Relationen die Fundiertheit erhalten bleibt. Hierbei ist es wichtig, dass die Größe der Tupel, die mit lexikographischen Relationen verglichen werden, festgelegt ist. Da $>_{alph}$ eine fundierte Relation ist, ist auch die Relation $(\succ_{alph})^n_{lex}$ zum Vergleich von Wörtern mit n Buchstaben fundiert. Die Relation, die in Lexika verwendet wird, ist hingegen nicht fundiert, da hier die Wörter beliebig lang werden können. Hier gilt z.B.

$$a > ba > bba > bbba > \ldots$$

Satz 4.4.5 (Fundiertheit bleibt bei lexikographischen Kombinationen erhalten) *Sei $\succ_1$ eine Relation auf der nicht-leeren Menge T_1 und $\succ_2$ eine Relation auf der nicht-leeren Menge T_2. Dann sind $\succ_1$ und $\succ_2$ fundiert gdw. ihre lexikographische Kombination $\succ_{1\times 2}$ fundiert ist.*

Beweis. Wir zeigen zuerst die "$\Leftarrow$"-Richtung. Sei $\succ_{1\times 2}$ fundiert. Falls $\succ_1$ nicht fundiert wäre, so gäbe es eine unendliche Folge $u_0 \succ_1 u_1 \succ_1 \ldots$ Sei $v \in T_2$ beliebig. Dann folgt $(u_0, v) \succ_{1\times 2} (u_1, v) \succ_{1\times 2} \ldots$ im Widerspruch zur Fundiertheit von $\succ_{1\times 2}$.

Wäre $\succ_2$ nicht fundiert, so gäbe es eine unendliche Folge $v_0 \succ_2 v_1 \succ_2 \ldots$ Mit $u \in T_1$ beliebig erhält man dann $(u, v_0) \succ_{1\times 2} (u, v_1) \succ_{1\times 2} \ldots$ im Widerspruch zur Fundiertheit von $\succ_{1\times 2}$.

Nun zeigen wir die "$\Rightarrow$"-Richtung, d.h., dass aus der Fundiertheit von $\succ_1$ und $\succ_2$ auch die Fundiertheit von $\succ_{1\times 2}$ folgt. Falls es eine unendliche Folge $(u_0, v_0) \succ_{1\times 2} (u_1, v_1) \succ_{1\times 2} \ldots$ gäbe, so würde $u_0 \succeq_1 u_1 \succeq_1 \ldots$ gelten, wobei $\succeq_1$ die reflexive Hülle von $\succ_1$ bezeichnet. Da $\succ_1$ fundiert ist, kann diese Folge nur endlich viele $\succ_1$-Schritte besitzen. Es existiert also ein $i \in \mathbb{N}$ mit $u_j = u_{j+1}$ für alle $j \geq i$. Damit folgt $v_i \succ_2 v_{i+1} \succ_2 \ldots$ im Widerspruch zur Fundiertheit von $\succ_2$. □

Der obige Satz garantiert, dass wir eine fundierte Ordnung $\succ$ auf Termen zu einer fundierten Ordnung $\succ_{lex}^n$ auf n-Tupeln von Termen erweitern können. Dies werden wir nun benutzen, um eine Ordnung für Terminierungsbeweise zu entwickeln, die der Einbettungsordnung deutlich überlegen ist.

Die Terminierung des plus-TES lässt sich nicht mit der Einbettungsordnung zeigen, da $\mathsf{plus}(\mathsf{succ}(x), y) \not\succ_{emb} \mathsf{succ}(\mathsf{plus}(x, y))$ gilt. Die Schwäche der Einbettungsordnung liegt also daran, dass sie Terme mit unterschiedlichen äußeren Funktionssymbolen (wie plus und succ) nur dann miteinander vergleichen kann, wenn der eine Term bereits kleiner oder gleich einem echten Teilterm des anderen ist. Auch für die im Folgenden vorgestellten Relationen reicht es für $f(s_1, \ldots, s_n) \succ t$ immer, wenn man zeigen kann, dass $s_i \succeq t$ für ein $i \in \{1, \ldots, n\}$ gilt. Man muss aber die Einbettungsordnung nun so erweitern, dass auch ansonsten ein Vergleich von Termen $f(s_1, \ldots, s_n)$ und $g(t_1, \ldots, t_m)$ möglich ist.

Die Hauptidee der im Folgenden vorgestellten Ordnungen ist, Terme aufgrund ihres ersten Funktionssymbols zu vergleichen und dann auf rekursive Weise anschließend ihre Teilterme zu vergleichen. Die Idee hierzu ist daher, dass man auch eine Ordnung $\sqsupset$ auf Funktionssymbolen einführt. Die Ordnung $\sqsupset$ wird oft *Präzedenz* genannt und gibt die Priorität der verschiedenen Funktionssymbole an. Ein Term $f(s_1, \ldots, s_n)$ wird nun als größer als der Term $g(t_1, \ldots, t_m)$ betrachtet (d.h., $f(s_1, \ldots, s_n) \succ g(t_1, \ldots, t_m)$), wenn $f \sqsupset g$ ist. Allerdings muss man zusätzlich dann auch noch fordern, dass der Gesamtterm $f(s_1, \ldots, s_n)$ zumindest auch größer als die Teilterme $t_1, \ldots, t_m$ ist (d.h., $f(s_1, \ldots, s_n) \succ t_j$ für alle $j \in \{1, \ldots, m\}$). Der Grund ist, dass man ansonsten keine fundierte Relation erhält, da dann sowohl $f(x) \succ g(f(x))$ (wegen $f \sqsupset g$) als auch $g(f(x)) \succ f(x)$ gelten würde (da $f(x)$ ein echter Teilterm von $g(f(x))$ ist).

Wenn wir zwei Terme $f(s_1, \ldots, s_n)$ und $f(t_1, \ldots, t_n)$ mit dem gleichen Funktionssymbol vergleichen wollen, wurde in der Einbettungsordnung $s_i \succ t_i$ für ein i und $s_j \succeq t_j$ für alle $j \neq i$ gefordert. Die zweite Forderung lässt sich aber abschwächen, indem man stattdessen nun die Tupel $(s_1, \ldots, s_n)$ und $(t_1, \ldots, t_n)$ *lexikographisch* vergleicht. Man verlangt also nach wie vor $s_i \succ t_i$ für ein i, aber $s_j \succeq t_j$ wird nur für die j mit $1 \leq j < i$ verlangt. Beispielsweise wäre damit $f(\mathsf{succ}(\mathcal{O}), \mathcal{O}) \succ f(\mathcal{O}, \mathsf{succ}(\mathcal{O}))$. Man kann daher auch gleich $s_j = t_j$ für alle $1 \leq j < i$ verlangen (denn als i kann man einfach die erste Argumentstelle wählen, wo sich die beiden Terme unterscheiden).

Allerdings muss man wieder zusätzlich $f(s_1, \ldots, s_n) \succ t_j$ für alle $j \in \{1, \ldots, n\}$ verlangen (für $1 \leq j \leq i$ gilt dies natürlich trivialerweise). Der Grund ist wieder, dass man ansonsten keine fundierte Relation erhält, da dann sowohl $f(\mathsf{succ}(\mathcal{O}), \mathcal{O}) \succ f(\mathcal{O}, f(\mathsf{succ}(\mathcal{O}), \mathcal{O}))$ als auch $f(\mathcal{O}, f(\mathsf{succ}(\mathcal{O}), \mathcal{O})) \succ f(\mathsf{succ}(\mathcal{O}), \mathcal{O})$ gelten würde (da $f(\mathsf{succ}(\mathcal{O}), \mathcal{O})$ ein echter Teilterm von $f(\mathcal{O}, f(\mathsf{succ}(\mathcal{O}), \mathcal{O}))$ ist).[4] Dies führt zur lexikographischen Pfadordnung.

Definition 4.4.6 (Lexikographische Pfadordnung [KL80]) *Für eine Signatur Σ und eine fundierte Ordnung ("*Präzedenz*") $\sqsupset$ über Σ ist die* lexikographische Pfadordnung (LPO) $\succ_{lpo}$ *über $\mathcal{T}(\Sigma, \mathcal{V})$ definiert als $s \succ_{lpo} t$ gdw.*

- $s = f(s_1, \ldots, s_n)$ *und* $s_i \succeq_{lpo} t$ *für ein* $i \in \{1, \ldots, n\}$ *oder*
- $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $f \sqsupset g$ *und* $f(s_1, \ldots, s_n) \succ_{lpo} t_j$ *für alle* $j \in \{1, \ldots, m\}$ *oder*
- $s = f(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n)$, $t = f(s_1, \ldots, s_{i-1}, t_i, t_{i+1}, \ldots, t_n)$, $s_i \succ_{lpo} t_i$ *und* $s = f(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n) \succ_{lpo} t_j$ *für alle* $j \in \{i+1, \ldots, n\}$.

[4] Man beachte, dass man nicht stattdessen $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, $s_i \succ t_i$ für ein i und $s \succ t_j$ für alle $j \neq i$ verlangen kann. Das Problem ist, dass man auf diese Weise $f(\mathcal{O}, \mathsf{succ}(\mathcal{O})) \succ f(\mathsf{succ}(\mathcal{O}), \mathcal{O}) \succ f(\mathcal{O}, \mathsf{succ}(\mathcal{O}))$ hätte.

Hierbei bezeichnet $\succeq_{lpo}$ die reflexive Hülle von $\succ_{lpo}$, d.h., die Relation mit $s \succeq_{lpo} t$ gdw. $s \succ_{lpo} t$ oder $s = t$.

Der dritte Fall der obigen Definition lässt sich auch wie folgt schreiben:

- $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, $(s_1, \ldots, s_n)$ $(\succ_{lpo})^n_{lex}$ $(t_1, \ldots, t_n)$ und $s \succ_{lpo} t_j$ für alle $j \in \{1, \ldots, n\}$.

Man erkennt also, dass man in der Tat die Argumente von gleichen Funktionssymbolen auf lexikographische Weise miteinander vergleicht.

Die lexikographische Pfadordnung lässt sich ebenfalls für Terminierungsbeweise verwenden, denn da die verwendete Präzedenz $\sqsupset$ fundiert ist, handelt es sich wieder um eine Reduktionsordnung. Die Fundiertheit der Präzedenz ist hierbei natürlich notwendig, um die Fundiertheit der lexikographischen Pfadordnung zu garantieren. (Da wir nur endliche Signaturen betrachten und da $\sqsupset$ transitiv ist, ist die Fundiertheit von $\sqsupset$ gleichbedeutend zu der Irreflexivität von $\sqsupset$, die sich natürlich leicht überprüfen lässt.) Bevor wir zeigen, dass es sich in der Tat um eine Simplifikations- und damit um eine Reduktionsordnung handelt, wollen wir die lexikographische Pfadordnung zuerst mit zwei Beispielen illustrieren.

Beispiel 4.4.7 Mit der lexikographischen Pfadordnung kann man die Terminierung vieler Termersetzungssysteme und Programme zeigen. Betrachten wir das TES mit den Regeln für plus und times.

$$\begin{aligned}
\mathsf{plus}(\mathcal{O}, y) &\to y \\
\mathsf{plus}(\mathsf{succ}(x), y) &\to \mathsf{succ}(\mathsf{plus}(x, y)) \\
\mathsf{times}(\mathcal{O}, y) &\to \mathcal{O} \\
\mathsf{times}(\mathsf{succ}(x), y) &\to \mathsf{plus}(y, \mathsf{times}(x, y))
\end{aligned}$$

Um seine Terminierung zu beweisen, muss eine Präzedenz $\sqsupset$ gefunden werden, so dass die folgenden Ungleichungen für die zugehörige lexikographische Pfadordnung erfüllt sind.

$$\mathsf{plus}(\mathcal{O}, y) \succ_{lpo} y \tag{4.8}$$
$$\mathsf{plus}(\mathsf{succ}(x), y) \succ_{lpo} \mathsf{succ}(\mathsf{plus}(x, y)) \tag{4.9}$$
$$\mathsf{times}(\mathcal{O}, y) \succ_{lpo} \mathcal{O} \tag{4.10}$$
$$\mathsf{times}(\mathsf{succ}(x), y) \succ_{lpo} \mathsf{plus}(y, \mathsf{times}(x, y)) \tag{4.11}$$

Die erste und die dritte Ungleichung (4.8) und (4.10) gelten bereits bei der Einbettungsordnung und somit bei jeder lexikographischen Pfadordnung (denn wir werden in Satz 4.4.9 zeigen, dass jede lexikographische Pfadordnung auch eine Simplifikationsordnung ist). Die zweite Ungleichung (4.9) kann nur gelten, falls $\mathsf{plus} \sqsupset \mathsf{succ}$ ist. Dann reicht es, zu zeigen, dass $\mathsf{plus}(\mathsf{succ}(x), y) \succ_{lpo} \mathsf{plus}(x, y)$ gilt. Dies folgt jedoch sofort aus $\mathsf{succ}(x) \succ_{lpo} x$ und $\mathsf{plus}(\mathsf{succ}(x), y) \succ_{lpo} y$. Die letzte Ungleichung (4.11) erzwingt die Präzedenz $\mathsf{times} \sqsupset \mathsf{plus}$. Dann ist nur noch $\mathsf{times}(\mathsf{succ}(x), y) \succ_{lpo} y$ und $\mathsf{times}(\mathsf{succ}(x), y) \succ_{lpo} \mathsf{times}(x, y)$ zu zeigen, was offensichtlich erfüllt ist. Wenn man eine Präzedenz mit $\mathsf{times} \sqsupset \mathsf{plus} \sqsupset \mathsf{succ}$ wählt, sind also die vier Ungleichungen (4.8) - (4.11) erfüllt und damit terminiert das Termersetzungssystem.

Beispiel 4.4.8 Um die Arbeitsweise des lexikographischen Vergleichs zu verdeutlichen, betrachten wir als weiteres Beispiel ein TES mit einem modifizierten Additionsalgorithmus.

$$\begin{aligned} \mathsf{sum}(\mathcal{O}, y) &\to y \\ \mathsf{sum}(\mathsf{succ}(x), y) &\to \mathsf{sum}(x, \mathsf{succ}(y)) \end{aligned}$$

Für seine Terminierung sind die folgenden Ungleichungen nachzuweisen:

$$\begin{aligned} \mathsf{sum}(\mathcal{O}, y) &\succ_{lpo} y \\ \mathsf{sum}(\mathsf{succ}(x), y) &\succ_{lpo} \mathsf{sum}(x, \mathsf{succ}(y)) \end{aligned}$$

Man erkennt, dass diese Ungleichungen bei der Präzedenz $\mathsf{sum} \sqsupset \mathsf{succ}$ erfüllt sind und dass sum deshalb terminiert. Die erste Ungleichung gilt bereits bei der Einbettungsordnung und bei der zweiten Ungleichung wird das erste Argument kleiner ($\mathsf{succ}(x) \succ_{lpo} x$) und das zweite Argument $\mathsf{succ}(y)$ auf der rechten Seite ist zumindest kleiner als die gesamte linke Seite (d.h., $\mathsf{sum}(\mathsf{succ}(x), y) \succ_{lpo} \mathsf{succ}(y)$). Hier müssen die Argumente von sum wirklich mit einer lexikographischen Ordnung verglichen werden, denn während das erste Argument kleiner wird (aus $\mathsf{succ}(x)$ wird x) wird das zweite Argument größer (aus y wird $\mathsf{succ}(y)$).

Nun zeigen wir den Satz, der sicher stellt, dass man die lexikographische Pfadordnung tatsächlich für Terminierungsbeweise verwenden kann. Hierbei ist wieder wichtig, dass wir nur endliche Signaturen betrachten. Zum Beweis verwenden wir den Satz von Kruskal, d.h., wir zeigen nur, dass $\succ_{lpo}$ stabil, monoton, transitiv und irreflexiv ist und außerdem die Teiltermeigenschaft erfüllt. Aufgrund des Satzes von Kruskal (Satz 4.4.2) folgt daraus auch die Fundiertheit.

Satz 4.4.9 (Eigenschaften der lexikographischen Pfadordnung) *Die lexikographische Pfadordnung $\succ_{lpo}$ ist eine Simplifikations- und somit eine Reduktionsordnung.*

Beweis. Wir zeigen die Teiltermeigenschaft, Monotonie, Stabilität, Transitivität und Irreflexivität von $\succ_{lpo}$. Daraus folgt dann gemäß Satz 4.4.2, dass $\succ_{lpo}$ eine Simplifikations- und Reduktionsordnung ist.

Teiltermeigenschaft

Es gilt $x_i \succeq_{lpo} x_i$ und somit $f(x_1, \ldots, x_n) \succ_{lpo} x_i$ für alle $f \in \Sigma$ und alle $i \in \{1, \ldots, n\}$ aufgrund der ersten Bedingung in Def. 4.4.6.

Monotonie

Wir zeigen folgende Aussage:

$$\text{Falls } s \succ_{lpo} t, \text{ dann } q[s]_\pi \succ_{lpo} q[t]_\pi.$$

Hierzu verwenden wir strukturelle Induktion über π. Im Fall $\pi = \epsilon$ ist die Aussage trivial. Im Fall $\pi = i\pi'$ gilt $q[s]_\pi = f(q_1, \ldots, q_i[s]_{\pi'}, \ldots, q_n)$ und $q[t]_\pi = f(q_1, \ldots, q_i[t]_{\pi'}, \ldots, q_n)$. Aufgrund der Induktionshypothese folgt $q_i[s]_{\pi'} \succ_{lpo} q_i[t]_{\pi'}$. Die erste Bedingung in Def. 4.4.6 impliziert $f(q_1, \ldots, q_i[s]_{\pi'}, \ldots, q_n) \succ_{lpo} q_j$ für alle $j \in \{i+1, \ldots, n\}$. Aus der dritten Bedingung in Def. 4.4.6 ergibt sich also $q[s]_\pi = f(q_1, \ldots, q_i[s]_{\pi'}, \ldots, q_n) \succ_{lpo} f(q_1, \ldots, q_i[t]_{\pi'}, \ldots, q_n) = q[t]_\pi$.

Stabilität

Wir beweisen die folgende Aussage:

$$\text{Falls } s \succ_{lpo} t, \text{ dann } s\sigma \succ_{lpo} t\sigma$$

Hierzu verwenden wir noethersche Induktion über $\rhd^2_{lex}$. Wenn wir die Aussage für s und t zeigen wollen, können wir sie also für alle s', t' voraussetzen, bei denen $(s,t) \rhd^2_{lex} (s',t')$ gilt (d.h., für (s',t') mit $s \rhd s'$ und für (s,t') mit $t \rhd t'$). Eine noethersche Induktion über diese Relation ist möglich, da $\rhd^2_{lex}$ nach Satz 4.4.5 fundiert ist. Wir betrachten die einzelnen Fälle von Def. 4.4.6.

Falls $s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{lpo} t$, so folgt aus der Induktionshypothese $s_i\sigma \succeq_{lpo} t\sigma$ und damit $s\sigma = f(s_1\sigma, \ldots, s_n\sigma) \succ_{lpo} t\sigma$.

Falls $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $f \sqsupset g$ und $s \succ_{lpo} t_j$ für alle $j \in \{1, \ldots, m\}$, so folgt aufgrund der Induktionshypothese $s\sigma \succ_{lpo} t_j\sigma$ und daher $s\sigma = f(s_1\sigma, \ldots, s_n\sigma) \succ_{lpo} g(t_1\sigma, \ldots, t_m\sigma) = t\sigma$.

Falls schließlich $s = f(s_1, \ldots, s_i, \ldots, s_n)$, $t = f(s_1, \ldots, t_i, \ldots, t_n)$, $s_i \succ_{lpo} t_i$ und $s \succ_{lpo} t_j$ für alle $j \in \{i+1, \ldots, n\}$ ist, so folgt aufgrund der Induktionshypothese $s_i\sigma \succ_{lpo} t_i\sigma$ und jeweils $s\sigma \succ_{lpo} t_j\sigma$. Damit erhält man wiederum $s\sigma \succ_{lpo} t\sigma$.

Transitivität

Wir beweisen die folgende Aussage:

$$\text{Falls } s \succ_{lpo} t \text{ und } t \succ_{lpo} r, \text{ dann } s \succ_{lpo} r$$

Hierzu verwenden wir noethersche Induktion über $\rhd^3_{lex}$. Wenn wir die Aussage für s, t und r zeigen wollen, dürfen wir sie also für alle s', t', r' mit $(s,t,r) \rhd^3_{lex} (s',t',r')$ voraussetzen. Die Fundiertheit der Induktionsrelation folgt wieder aus Satz 4.4.5. Wir betrachten die einzelnen Fälle von Def. 4.4.6.

Zunächst untersuchen wir den Fall, dass eine der beiden Ungleichungen $s \succ_{lpo} t$ oder $t \succ_{lpo} r$ aufgrund der ersten Bedingung in Def. 4.4.6 gilt.

- Sei $s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{lpo} t$ für ein $i \in \{1, \ldots, n\}$. Aus $s_i \succeq_{lpo} t \succ_{lpo} r$ folgt aufgrund der Induktionshypothese $s_i \succ_{lpo} r$. Damit erhält man auch $s \succ_{lpo} r$.
- Sei nun $t = g(t_1, \ldots, t_m)$ und $t_j \succeq_{lpo} r$ für ein $j \in \{1, \ldots, m\}$.
  Außerdem sei $s \succ_{lpo} t$ nicht mit Hilfe der ersten Bedingung aus Def. 4.4.6 entstanden. Daraus ergibt sich $s \succ_{lpo} t_j$.
  Insgesamt erhält man also $s \succ_{lpo} t_j \succeq_{lpo} r$, was aufgrund der Induktionshypothese $s \succ_{lpo} r$ ergibt.

Somit verbleiben nun die Fälle, bei denen $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$ und $r = h(r_1, \ldots, r_l)$ ist, wobei $f \sqsupseteq g \sqsupseteq h$ gilt. Aufgrund der Transitivität von $\sqsupset$ (denn $\sqsupset$ ist eine Ordnung), ist somit $f \sqsupseteq h$. Außerdem gilt $t \succ_{lpo} r_k$ für alle $k \in \{1, \ldots, l\}$.

Falls $f \sqsupset h$ ist (d.h., für $s \succ_{lpo} t$ oder $t \succ_{lpo} r$ wurde die zweite Bedingung verwendet), so folgt aus $s \succ_{lpo} t \succ_{lpo} r_k$ aufgrund der Induktionshypothese $s \succ_{lpo} r_k$ für alle $k \in \{1, \ldots, l\}$. Mit der zweiten Bedingung von Def. 4.4.6 ergibt sich $s \succ_{lpo} r$.

Ansonsten gilt $f = g = h$ und sowohl für $s \succ_{lpo} t$ als auch für $t \succ_{lpo} r$ wurde die dritte Bedingung verwendet (aufgrund der Fundiertheit von $\succ$). Es gibt somit ein i mit $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$ und $s_i \succ_{lpo} t_i$ und ein j mit $t_1 = r_1, \ldots, t_{j-1} = r_{j-1}$ und $t_j \succ_{lpo} r_j$. Sei $d = \min(i, j)$. Dann gilt $s_1 = t_1 = r_1, \ldots, s_{d-1} = t_{d-1} = r_{d-1}$. Falls $d = i < j$ ist, so gilt $s_d \succ_{lpo} t_d = r_d$ und wenn $d = j < i$ ist, so gilt $s_d = t_d \succ_{lpo} r_d$. Falls schließlich $d = i = j$ ist, so ergibt sich $s_d \succ_{lpo} t_d \succ_{lpo} r_d$ und aufgrund der Induktionshypothese $s_d \succ_{lpo} r_d$. Außerdem gilt $s \succ_{lpo} t \succ_{lpo} r_k$ und damit aufgrund der Induktionshypothese $s \succ_{lpo} r_k$ für alle $k \in \{1, \ldots, l\}$. Daher gilt aufgrund der dritten Bedingung auch $s \succ_{lpo} r$.

Irreflexivität

Wir zeigen, dass für alle Terme s jeweils $s \not\succ_{lpo} s$ gilt. Hierzu verwenden wir strukturelle Induktion über den Term s. Wir betrachten die einzelnen Fälle von Def. 4.4.6.

Wir nehmen zuerst an, dass $s = f(s_1, \ldots, s_n)$ und $s_i \succeq_{lpo} s$ gilt. Aufgrund der Teiltermeigenschaft und der Stabilität gilt aber auch $s \succ_{lpo} s_i$, wodurch sich aufgrund der Transitivität $s_i \succ_{lpo} s_i$ im Widerspruch zur Induktionshypothese ergibt.

Der zweite Fall der Definition 4.4.6 kann nicht verwendet werden, um $s \succ_{lpo} s$ zu erhalten, da ansonsten für das oberste Funktionssymbol f von s die Beziehung $f \sqsupset f$ gelten würde, was der Fundiertheit von $\sqsupset$ widerspricht.

Würde $s \succ_{lpo} s$ mit dem dritten Fall der Definition gelten, so müsste es einen Teilterm s_i von s geben mit $s_i \succ_{lpo} s_i$. Dies widerspricht aber der Induktionshypothese. □

Es ist entscheidbar, ob (bei festgelegter Präzedenz $\sqsupset$) $s \succ_{lpo} t$ für zwei Terme s und t gilt. Hierzu kann man (wie bei der Einbettungsordnung) ein rekursives Programm angeben, das einfach die Bedingungen von Def. 4.4.6 abarbeitet.

Für ein TES $\mathcal{R}$ ist es auch entscheidbar, ob es eine Präzedenz $\sqsupset$ auf den Funktionssymbolen gibt, so dass für die dazugehörige lexikographische Pfadordnung $l \succ_{lpo} r$ für alle Regeln $l \to r \in \mathcal{R}$ gilt. (Allerdings ist dies ein NP-vollständiges Problem [KN85].) Der Grund ist, dass wir nur endliche Signaturen Σ betrachten, so dass alle Möglichkeiten für die Definition der Präzedenz probiert werden können. In der Praxis verwendet man natürlich effizientere Algorithmen, die mit einer völlig unbestimmten Präzedenz $\sqsupset$ beginnen und sukzessive die Regeln untersuchen. Nur wenn es eine Regel $l \to r$ gibt, bei der $l \succ_{lpo} r$ nur dann gilt, wenn ein bestimmtes Funktionssymbol f größer als ein anderes Symbol g ist, erweitert man die bislang festgelegte Präzedenz so, dass $f \sqsupset g$ gilt. Die benötigte Präzedenz wird sozusagen "on demand" ergänzt. Selbstverständlich muss man hierbei darauf achten, dass sich keine Zyklen bei der Präzedenz ergeben, d.h., dass die Präzedenz fundiert bleibt. Dies lässt sich aber leicht automatisch überprüfen, da es ja nur endlich viele Funktionssymbole gibt. Wie die benötigte Präzedenz sukzessive bei der Bearbeitung der verschiedenen Ungleichungen festgelegt wird, wurde bereits in Bsp. 4.4.7 deutlich. Entsprechende Algorithmen finden sich beispielsweise in [DF85, CLM08].

Damit ergibt sich ein verbessertes Verfahren für automatische Terminierungsbeweise: Für ein TES $\mathcal{R}$ sucht man nach einer Präzedenz $\sqsupset$ auf der dazugehörigen Signatur, so dass bei der zugehörigen lexikographischen Pfadordnung die linken Seiten der definierenden Gleichungen jeweils größer als die entsprechenden rechten Seiten sind. Gelingt dies, so

terminiert das Programm (nach Satz 4.4.9 und Satz 4.3.4). Ansonsten kann man keine Aussage über die Terminierung von $\mathcal{R}$ treffen.

Dieses Verfahren ist deutlich stärker als die Verwendung der Einbettungsordnung. Der Grund ist, dass jede lexikographische Pfadordnung eine Simplifikationsordnung ist und daher die Einbettungsordnung in jeder lexikographischen Pfadordnung enthalten ist.

In Def. 4.4.6 findet der lexikographische Vergleich der Argumente immer von links nach rechts statt, d.h., wenn $s = f(s_1, \ldots, s_i, \ldots, s_n)$, $t = f(t_1, \ldots, t_i, \ldots, t_n)$ und $s_i \succ_{lpo} t_i$ ist, so verlangen wir $s_j = t_j$ für $1 \leq j < i$ und $s \succ_{lpo} t_j$ für $i < j \leq n$. Selbstverständlich könnte man stattdessen auch einen lexikographischen Vergleich von rechts nach links verwenden. Die dritte Bedingung in der Definition der lexikographischen Pfadordnung müsste dann wie folgt geändert werden:

- $s = f(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n)$, $t = f(t_1, \ldots, t_{i-1}, t_i, s_{i+1}, \ldots, s_n)$, $s_i \succ_{lpo} t_i$ und $s = f(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n) \succ_{lpo} t_j$ für alle $j \in \{1, \ldots, i-1\}$.

Diese Variante der lexikographischen Pfadordnung ist z.B. nötig, wenn man die Terminierung des folgenden TES nachweisen will.

$$\begin{aligned} \mathsf{pred}(\mathcal{O}) &\to \mathcal{O} \\ \mathsf{pred}(\mathsf{succ}(x)) &\to x \\ \mathsf{minus}(x, \mathcal{O}) &\to x \\ \mathsf{minus}(x, \mathsf{succ}(y)) &\to \mathsf{minus}(\mathsf{pred}(x), y) \end{aligned}$$

Die Ungleichungen, die aus den ersten drei Regeln entstehen, werden bereits von der Einbettungsordnung erfüllt. Für die letzte Regel müssen wir jedoch

$$\mathsf{minus}(x, \mathsf{succ}(y)) \succ_{lpo} \mathsf{minus}(\mathsf{pred}(x), y)$$

nachweisen. Wenn man die Argumente lexikographisch von links nach rechts vergleicht, ist dies nicht möglich, da $x \not\succeq_{lpo} \mathsf{pred}(x)$ gilt. Bei der Variante der lexikographischen Pfadordnung, die von rechts nach links vergleicht, gelingt dieser Beweis jedoch bei der Präzedenz $\mathsf{minus} \sqsupset \mathsf{pred}$, da wir dann $\mathsf{succ}(y) \succ_{lpo} y$ und $\mathsf{minus}(x, \mathsf{succ}(y)) \succ_{lpo} \mathsf{pred}(x)$ haben. Das obige Programm terminiert daher.

Neben dem lexikographischen Vergleich von links nach rechts und von rechts nach links sind natürlich auch alle anderen Permutationen der Argumente möglich. Um auch die Terminierung von Programmen nachzuweisen, die sowohl einen Algorithmus wie sum (für den man die Argumente von links nach rechts vergleichen muss) als auch einen Algorithmus wie minus enthalten (für den man von rechts nach links vorgehen muss), kann man jedes Funktionssymbol mit einem sogenannten *Status* versehen, der angibt, auf welche Weise die Argumente dieses Funktionssymbols verglichen werden sollen. Der Status von sum wäre dann die Permutation $\langle 1, 2 \rangle$ (da die Argumente in dieser Reihenfolge verglichen werden) und der Status von minus wäre $\langle 2, 1 \rangle$.

Es gibt jedoch auch TESe, bei denen es keine Permutation der Argumente gibt, so dass der lexikographische Vergleich erfolgreich ist.

Beispiel 4.4.10 Wir betrachten die folgende Variante der plus-TES, bei dem die Argumente im rekursiven Aufruf vertauscht wurden.

$$\begin{aligned} \mathsf{plus}(\mathcal{O}, y) &\to y \\ \mathsf{plus}(\mathsf{succ}(x), y) &\to \mathsf{succ}(\mathsf{plus}(y, x)) \end{aligned}$$

Für die zweite Regel lässt sich keine lexikographische Pfadordnung angeben, so dass die linke Seite größer als die rechte Seite wäre. Das Problem ist, dass weder das erste noch das zweite Argument von plus hierbei kleiner wird. Es gilt $\mathsf{succ}(x) \not\succ y$ und $y \not\succ x$ für jede fundierte und stabile Relation $\succ$. Das Problem liegt daran, dass in diesen Ungleichungen die rechte Seite Variablen enthält, die nicht auf der linken Seite auftreten.

Im obigen Beispiel besteht die Abhilfe darin, dass man die Argumente von plus nicht als Tupel, sondern als *Mengen* vergleicht. Dann spielt es keine Rolle mehr, an welcher Argumentposition ein Argument auftritt. Hierzu führen wir nun neben der lexikographischen Pfadordnung eine zweite Simplifikationsordnung für automatische Terminierungsbeweise mit TESen ein.

Im allgemeinen kann ein Funktionssymbol natürlich mehrere Argumente mit demselben Wert besitzen. Bei der Datenstruktur der Mengen werden jedoch doppelte Vorkommen eines Elements ignoriert. Es ist daher sinnvoll, statt Mengen sogenannte *Multimengen* zu verwenden. Multimengen unterscheiden sich von Mengen dadurch, dass Elemente *mehrfach* vorkommen können (wie bei linearen Listen). Im Unterschied zu Listen ist jedoch die Reihenfolge der Elemente unerheblich. Bei Multimengen gilt also $\{1, 1, 4\} = \{1, 4, 1\} \neq \{1, 4\}$.

Definition 4.4.11 (Multimenge) *Sei T eine nicht-leere Menge. Eine ungeordnete endliche Folge M von Elementen aus T heißt* Multimenge *über T. Wir bezeichnen die Menge aller (endlichen) Multimengen über T mit $\mathcal{M}(T)$.*

Eine solche Multimenge M ist durch ihre charakteristische Funktion $\#_M : T \to \mathbb{N}$ *definiert, wobei $\#_M(t)$ die Anzahl der Vorkommen des Elements t in der Multimenge M angibt. Es gilt also $t \in M$ gdw. $\#_M(t) > 0$. Auf Multimengen M, N werden nun die folgenden Operationen definiert:*

- *$M = N$ gdw. $\#_M(t) = \#_N(t)$ für alle $t \in T$*
- *$M \subseteq N$ gdw. $\#_M(t) \leq \#_N(t)$ für alle $t \in T$*
- *$M \subset N$ gdw. $M \subseteq N$ und $M \neq N$*
- *$M \cup N$ ist die Multimenge mit $\#_{M \cup N}(t) = \#_M(t) + \#_N(t)$ für alle $t \in T$*
- *$M \cap N$ ist die Multimenge mit $\#_{M \cap N}(t) = \min(\#_M(t), \#_N(t))$ für alle $t \in T$*
- *$M \setminus N$ ist die Multimenge mit $\#_{M \setminus N}(t) = \max(0, \#_M(t) - \#_N(t))$ für alle $t \in T$*

Beispiel 4.4.12 Sei $M, N \in \mathcal{M}(\mathbb{N})$ mit $M = \{1, 1, 4\}$ und $N = \{1, 2, 2, 4, 4, 5\}$. Dann gilt:

- $M \not\subseteq N$ und $N \not\subseteq M$
- $M \cup N = \{1, 1, 1, 2, 2, 4, 4, 4, 5\}$
- $M \cap N = \{1, 4\}$
- $M \setminus N = \{1\}$
- $N \setminus M = \{2, 2, 4, 5\}$

Jede Relation $\succ$ auf einer Menge T kann zu einer Relation $\succ_{mul}$ zwischen Multimengen von Elementen aus T erweitert werden. Die Idee hierbei ist, dass $M \succ_{mul} N$ gilt, falls N aus M entsteht, indem einige Elemente von M durch (beliebig viele) $\succ$-kleinere Elemente ersetzt werden.

Definition 4.4.13 (Multimengenrelation) *Sei $\succ$ eine Relation auf T, d.h., $\succ \subseteq T \times T$. Dann ist die Relation $\succ_{mul} \subseteq \mathcal{M}(T) \times \mathcal{M}(T)$ definiert als $M \succ_{mul} N$ gdw. es existieren $X, Y \in \mathcal{M}(T)$ mit*

- $\varnothing \neq X \subseteq M$
- $N = (M \setminus X) \cup Y$
- *für jedes $y \in Y$ existiert ein $x \in X$ mit $x \succ y$*

Beispiel 4.4.14 Für $M = \{3, 6, 8\}$ und $N = \{4, 5, 6, 6, 7, 7\}$ gilt $M(>_{\mathbb{N}})_{mul}N$, denn die Teilmenge $X = \{3, 8\}$ wird durch die Menge $Y = \{4, 5, 6, 7, 7\}$ ersetzt und dabei ist in der Tat jedes Element von Y kleiner als das Element 8 aus X.

Neben der Erweiterung von Relationen auf Tupel von Elementen durch die lexikographische Kombination haben wir nun auch die Möglichkeit, Relationen auf Multimengen von Elementen zu erweitern. Ebenso wie bei der lexikographischen Kombination garantiert auch das obige Konstruktionsprinzip zur Erweiterung von Relationen auf Multimengen, dass dabei die Fundiertheit erhalten bleibt.

Satz 4.4.15 (Fundiertheit bleibt bei Multimengen erhalten [DM79]) *Sei $\succ$ eine Relation auf einer Menge T. Dann gilt: $\succ$ ist fundiert gdw. $\succ_{mul}$ fundiert ist.*

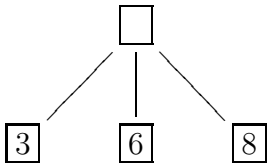
Beweis. Aus der Fundiertheit von $\succ_{mul}$ folgt offensichtlich die Fundiertheit von $\succ$. Gäbe es nämlich eine unendliche absteigende Folge $t_0 \succ t_1 \succ \ldots$, so hätte man nach Definition von $\succ_{mul}$ auch $\{t_0\} \succ_{mul} \{t_1\} \succ_{mul} \ldots$

Nun zeigen wir, dass aus der Fundiertheit von $\succ$ auch die Fundiertheit von $\succ_{mul}$ folgt. Wir nehmen hierzu an, dass es eine unendlich absteigende Folge $M_0 \succ_{mul} M_1 \succ_{mul} \ldots$ gibt. Aus dieser unendlichen Folge konstruieren wir nun einen unendlichen Baum, dessen Knoten mit Elementen von T markiert sind. Der Baum wird so definiert, dass die Elemente entlang eines Pfades bezüglich $\succ$ abnehmen, d.h., wenn ein Knoten s das (direkte) Kind t hat, dann

gilt $s \succ t$. Der Baum hat außerdem endlichen Verzweigungsgrad und somit folgt aus dem Lemma von König (Satz 4.1.3), dass er einen unendlichen Pfad besitzt. Dieser unendliche Pfad ergibt dann aber eine unendliche absteigende $\succ$-Folge, was der Fundiertheit von $\succ$ widerspricht.

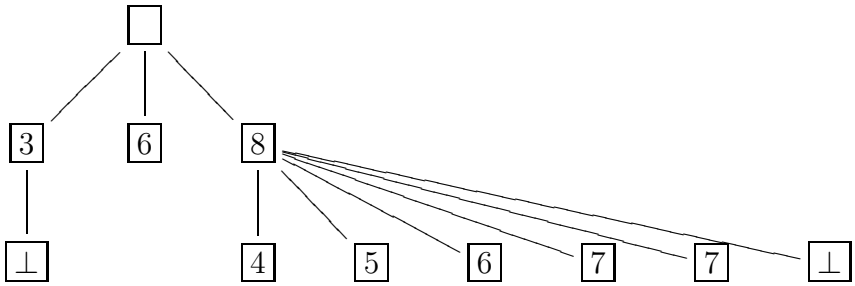
Wir müssen nun angeben, wie wir den unendlichen Baum erzeugen. Hierzu ist es hilfreich, bei den Markierungen der Knoten noch ein weiteres Objekt $\perp$ mit $\perp \notin T$ zuzulassen. Wir definieren also $T_\perp = T \cup \{\perp\}$ und markieren die Knoten des zu konstruierenden Baums mit Werten aus $T_\perp$. Auch die Relation $\succ$ wird auf $T_\perp$ erweitert, indem wir $t \succ \perp$ für alle $t \in T$ definieren. Damit bleibt die Fundiertheit von $\succ$ offensichtlich erhalten.

Nun geben wir an, wie man sukzessive den unendlichen Baum konstruiert. Im i-ten Schritt der Konstruktion sind die nicht mit $\perp$ markierten Blätter des Baums gerade mit den Elementen von M_i markiert. Der initiale Baum (im 0-ten Schritt) hat eine nicht markierte Wurzel und darunter einen Nachfolger für jedes Element aus M_0. Wenn also $M_0 = \{3, 6, 8\}$ ist, dann sieht der Baum wie folgt aus:



Da M_0 endlich ist, ist der Verzweigungsgrad des initialen Baums ebenfalls endlich.

Da $M_0 \succ_{mul} M_1$ gilt, gibt es endliche Multimengen X und Y wie in Def. 4.4.13. Für jedes $y \in Y$ fügen wir einen neuen Knoten mit der Markierung y hinzu und machen daraus ein Kind eines bisherigen Blatts $x \in X$ mit $x \succ y$. Nach Voraussetzung muss es ein solches $x \in X$ geben und nach der Konstruktion war der entsprechende Knoten bisher ein Blatt. Außerdem erhält jeder Knoten $x \in X$ ein weiteres Kind mit der Markierung $\perp$. Wenn $M_1 = \{4, 5, 6, 6, 7, 7\}$ ist, dann ergibt sich $X = \{3, 8\}$ und $Y = \{4, 5, 6, 7, 7\}$. Der Baum wird also wie folgt ergänzt:



Dieser Prozess wird dann für $M_2, M_3, \ldots$ unendlich oft wiederholt. Der Verzweigungsgrad bleibt dabei endlich, da alle M_i endlich sind. Die Anzahl der Knoten des Baums wächst aber in jedem Schritt (da mindestens ein $\perp$-Knoten angefügt wird, selbst dann, wenn $Y = \varnothing$ ist). Aufgrund des Lemmas von König gibt es also einen unendlichen Pfad in dem entstehenden Baum. Außerdem werden die Knoten auf jedem Pfad immer bezüglich $\succ$ kleiner (sofern man den Wurzelknoten ignoriert). Die Folge der Knotenmarkierungen auf dem unendlichen Pfad liefert daher eine unendlich absteigende $\succ$-Folge, was der Fundiertheit von

$\succ$ widerspricht. $\square$

Wenn wir zwei Terme $f(s_1, \ldots, s_n)$ und $f(t_1, \ldots, t_n)$ mit dem gleichen Funktionssymbol vergleichen, wurde in der Einbettungsordnung $s_i \succ t_i$ für ein i und $s_j \succeq t_j$ für alle $j \neq i$ gefordert. In der lexikographischen Pfadordnung wurden stattdessen die Argumenttupel $(s_1, \ldots, s_n)$ und $(t_1, \ldots, t_n)$ lexikographisch verglichen. Wir definieren nun die *rekursive Pfadordnung* (oder "Multimengenpfadordnung"), in der wir stattdessen die Multimenge der Argumente $\{s_1, \ldots, s_n\}$ mit der Multimenge $\{t_1, \ldots, t_n\}$ vergleichen.

Definition 4.4.16 (Rekursive Pfadordnung [Der82]) *Für eine Signatur Σ und eine fundierte Ordnung ("*Präzedenz*") $\sqsupset$ über Σ ist die* rekursive Pfadordnung (RPO) $\succ_{rpo}$ *über $\mathcal{T}(\Sigma, \mathcal{V})$ definiert als $s \succ_{rpo} t$ gdw.*

- $s = f(s_1, \ldots, s_n)$ *und* $s_i \succeq_{rpo} t$ *für ein* $i \in \{1, \ldots, n\}$ *oder*
- $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $f \sqsupset g$ *und* $f(s_1, \ldots, s_n) \succ_{rpo} t_j$ *für alle* $j \in \{1, \ldots, m\}$ *oder*
- $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$ *und* $\{s_1, \ldots, s_n\} \, (\succ_{rpo})_{mul} \, \{t_1, \ldots, t_n\}$.

Hierbei bezeichnet $\succeq_{rpo}$ die reflexive Hülle von $\succ_{rpo}$, d.h., die Relation mit $s \succeq_{rpo} t$ gdw. $s \succ_{rpo} t$ oder $s = t$.

Beispiel 4.4.17 Betrachten wir wieder das TES aus Bsp. 4.4.10.

$$\begin{array}{rcl} \mathsf{plus}(\mathcal{O}, y) & \to & y \\ \mathsf{plus}(\mathsf{succ}(x), y) & \to & \mathsf{succ}(\mathsf{plus}(y, x)) \end{array}$$

Mit der rekursiven Pfadordnung lässt sich nun zeigen, dass linke Seiten jeweils größer als die dazugehörigen rechten Seiten sind. Für die erste Regel ist dies trivial. Für die zweite Regel benötigt man die Präzedenz $\mathsf{plus} \sqsupset \mathsf{succ}$ und muss dann zeigen, dass $\mathsf{plus}(\mathsf{succ}(x), y) \succ_{rpo} \mathsf{plus}(y, x)$ ist. Dies bedeutet, dass $\{\mathsf{succ}(x), y\} \, (\succ_{rpo})_{mul} \, \{y, x\}$ bewiesen werden muss. Hier wurde die Teilmenge $X = \{\mathsf{succ}(x)\}$ der ersten Multimenge durch die Teilmenge $Y = \{x\}$ in der zweiten Multimenge ersetzt. Nach der Definition der Multimengenordnung muss man also nachweisen, dass es für jedes Element aus Y ein dazu größeres in X gibt. Da offensichtlich $\mathsf{succ}(x) \succ_{rpo} x$ gilt, ist damit $\mathsf{plus}(\mathsf{succ}(x), y) \succ_{rpo} \mathsf{plus}(y, x)$ bewiesen.

Die rekursive Pfadordnung lässt sich ebenfalls zu Terminierungsbeweisen verwenden, denn da die verwendete Präzedenz $\sqsupset$ eine fundierte Ordnung ist, handelt es sich wieder um eine Simplifikations- und damit eine Reduktionsordnung. Der Beweis davon ist ähnlich zum Beweis von Satz 4.4.9.

Satz 4.4.18 (Eigenschaften der rekursiven Pfadordnung) *Die rekursive Pfadordnung $\succ_{rpo}$ ist eine Simplifikations- und somit eine Reduktionsordnung.*

Bsp. 4.4.17 zeigt, dass es TESe gibt, bei denen der Terminierungsbeweis mit der rekursiven Pfadordnung geführt werden kann, wohingegen der Beweis mit der lexikographischen Pfadordnung scheitert. Umgekehrt gibt es aber auch viele Beispiele, wo man die lexikographische Pfadordnung benötigt und die rekursive Pfadordnung nicht erfolgreich ist.

Beispiel 4.4.19 Betrachten wir noch einmal das Additions-TES aus Bsp. 4.4.8.

$$\begin{array}{rcl} \mathsf{sum}(\mathcal{O}, y) & \to & y \\ \mathsf{sum}(\mathsf{succ}(x), y) & \to & \mathsf{sum}(x, \mathsf{succ}(y)) \end{array}$$

In Bsp. 4.4.8 wurde die Terminierung mit Hilfe der lexikographischen Pfadordnung gezeigt. Hingegen gelingt der Beweis mit der rekursiven Pfadordnung nicht. Für die zweite Regel müsste man $\mathsf{sum}(\mathsf{succ}(x), y) \succ_{rpo} \mathsf{sum}(x, \mathsf{succ}(y))$ und daher $\{\mathsf{succ}(x), y\}\,(\succ_{rpo})_{mul}\,\{x, \mathsf{succ}(y)\}$ zeigen. In diesem Fall wäre X die gesamte erste und Y die gesamte zweite Multimenge. Es gibt aber nicht zu jedem Element der zweiten Multimenge ein größeres Element in der ersten Multimenge, denn es gilt sowohl $\mathsf{succ}(x) \not\succ_{rpo} \mathsf{succ}(y)$ als auch $y \not\succ_{rpo} \mathsf{succ}(y)$.

Wie bei der lexikographischen Pfadordnung erwähnt, kann man jedes Funktionssymbol mit einem Status versehen, der angibt, in welcher Reihenfolge seine Argumente lexikographisch verglichen werden sollen. Das Konzept des Status kann nun so erweitert werden, dass man die lexikographische und die rekursive Pfadordnung verbindet. Dann gibt es neben den lexikographischen Status-Möglichkeiten nun auch die Möglichkeit, dass der Status angibt, dass die Argumente eines Funktionssymbols als Multimenge zu vergleichen sind. Die so entstehenden Ordnungen bezeichnet man als *rekursive Pfadordnungen mit Status (RPOS)* $\succ_{rpos}$. All diese Ordnungen sind wieder Reduktionsordnungen, die die Einbettungsordnung enthalten (d.h., es handelt sich um Simplifikationsordnungen) und es lässt sich automatisch überprüfen, ob es eine Präzedenz und eine geeignete Wahl für den Status der Funktionssymbole gibt, so dass $l \succ_{rpos} r$ für alle Regeln $l \to r$ gilt, vgl. [Der87, Ste95]. Hiermit erhält man also ein noch mächtigeres automatisches Terminierungsverfahren. Die Terminierung des TES, das sowohl die sum-Regeln aus Bsp. 4.4.19 als auch die plus-Regeln aus Bsp. 4.4.17 enthält, lässt sich beispielsweise weder mit der lexikographischen noch mit der rekursiven Pfadordnung alleine beweisen. Verwendet man hingegen eine rekursive Pfadordnung mit Status, wobei man die Argumente von sum lexikographisch von links nach rechts und die Argumente von plus als Multimenge vergleicht, so kann man die Terminierung leicht (automatisch) zeigen.

Die Frage, ob es eine rekursive Pfadordnung mit Status gibt, mit der man die Terminierung eines gegebenen TES nachweisen kann, ist entscheidbar. Dieses automatische Terminierungsverfahren lässt sich nicht nur für TESe verwenden, sondern es lässt sich direkt bei funktionalen Programmiersprachen einsetzen und auch auf andere (z.B. imperative) Programmiersprachen erweitern [BG99]. Da die Terminierung von TESen aber unentscheidbar ist, folgt damit sofort, dass es sich hierbei nur um ein unvollständiges Verfahren handelt. Es gibt also TESe, die zwar terminieren, deren Terminierung aber nicht mit rekursiven Pfadordnungen nachgewiesen werden kann. Ein Beispiel hierfür ist das folgende System.

$$\begin{array}{rcl} \mathsf{plus}(\mathsf{succ}(\mathsf{succ}(x)), \mathsf{succ}(y)) & \to & \mathsf{succ}(\mathsf{plus}(x, \mathsf{succ}(\mathsf{succ}(y)))) \\ \mathsf{plus}(\mathsf{succ}(x), \mathsf{succ}(\mathsf{succ}(y))) & \to & \mathsf{succ}(\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(x)), y)) \end{array}$$

Neben solchen rekursiven Pfadordnungen, wie sie in diesem Abschnitt vorgestellt wurden, gibt es aber auch noch andere Arten von Simplifikationsordnungen, die ebenfalls für automatische Terminierungsbeweise eingesetzt werden (z.B. die *Knuth-Bendix Ordnung*

[KB70] oder sogenannte *Polynomordnungen* [Lan79, Gie95]). Mit Polynomordnungen könnte man beispielsweise die Terminierung des obigen Systems nachweisen.

Die Frage, ob die Terminierung mit Hilfe einer Simplifikationsordnung gezeigt werden kann, ist unentscheidbar [MG95]. Außerdem gibt es darüber hinaus TESe, die zwar terminieren, bei denen dies jedoch mit keiner Simplifikationsordnung gezeigt werden kann. Ein einfaches solches System ist folgendes bekannte Beispiel [Der87].

$$\mathsf{f}(\mathsf{f}(x)) \;\to\; \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$$

Das TES terminiert, da in jedem Reduktionsschritt die Anzahl der nebeneinander stehenden f-Symbole verringert wird. Mit einer Simplifikationsordnung lässt sich die Terminierung aber nicht zeigen. Für jede Simplifikationsordnung $\succ$ gilt nämlich $\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \succ \mathsf{f}(\mathsf{f}(x))$ und somit $\mathsf{f}(\mathsf{f}(x)) \not\succ \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$ aufgrund der Fundiertheit von $\succ$.

TESe, deren Terminierung nicht mit Simplifikationsordnungen gezeigt werden können, sind nicht nur artifizielle Beispiele wie oben, sondern solche TESe treten häufig in der Praxis auf. Daher gibt es weitere Techniken zum automatischen Terminierungsbeweis (wie z.B. *Dependency Pairs* [AG00]), die auf den bislang vorgestellten Verfahren aufbauen und auch die Terminierung vieler solcher TESe nachweisen können. Für weitere Details hierzu wird auf die Literatur bzw. auf Vorlesungen oder Bücher über automatisierte Programmverifikation [Gie03] verwiesen.

Kapitel 5

Konfluenz von Termersetzungssystemen

Wie in Abbildung 3.1 illustriert, benötigen wir neben Techniken zur Überprüfung der Terminierung auch ein Verfahren, um die Konfluenz von Termersetzungssystemen zu untersuchen. Allgemein sind solche Techniken von Interesse, weil sie verwendet werden können, um Programme oder Berechnungsformalismen auf ihre Eindeutigkeit zu untersuchen.

Ebenso wie die Terminierung ist auch die Frage der Konfluenz im allgemeinen unentscheidbar. Für terminierende TESe ist es aber entscheidbar, ob sie konfluent sind. Sofern wir also bereits die Terminierung eines TES (z.B. mit den Techniken des vorigen Kapitels) nachgewiesen haben, so lernen wir nun eine Methode kennen, die automatisch herausfinden kann, ob das TES konfluent ist. In unserem Vorgehen zur Konstruktion konvergenter TESe (Abb. 3.1) wird immer zuerst die Terminierung sicher gestellt, so dass die Frage der Konfluenz hier tatsächlich entschieden werden kann.

Das Verfahren zur Untersuchung der Konfluenz (ebenso wie das Vervollständigungsverfahren im nächsten Kapitel) beruht auf der Verwendung der *Unifikation*. Darüber hinaus ist Unifikation von Termen auch in vielen weiteren Bereichen (außerhalb von Termersetzungssystemen) von großer Bedeutung. Wir stellen daher in Abschnitt 5.1 zuerst einen Unifikationsalgorithmus vor. Hierbei gehen wir auch darauf ein, wie dieser Algorithmus als Matchingalgorithmus verwendet werden kann. Matching von Termen wird natürlich immer benötigt, um die Termersetzungsrelation zu implementieren. Anschließend betrachten wir dann in Abschnitt 5.2 das Verfahren zur Konfluenzuntersuchung (bei terminierenden TESen). Schließlich zeigen wir in Abschnitt 5.3 ein hinreichendes Kriterium, um auch bei nicht-terminierenden TESen die Konfluenz nachzuweisen. Dieses Kriterium wird insbesondere in der funktionalen Programmierung verwendet.

5.1 Unifikation

Die Frage der *Unifikation* untersucht, ob es zu zwei Termen s und t eine Substitution σ gibt, so dass $s\sigma = t\sigma$ gilt. Bislang haben wir immer die *Gültigkeit* von Gleichungen in Algebren (bzw. in einer Menge von Algebren) untersucht. Dabei bedeutete $A \models s \equiv t$, dass *für alle* Variablenbelegungen die Terme s und t in der Algebra A gleich gedeutet

werden. Ebenso bedeutete das Wortproblem $s \equiv_{\mathcal{E}} t$, dass in allen Modellen A von $\mathcal{E}$ bei allen Variablenbelegungen s und t gleich interpretiert werden. Bislang haben wir also die Variablen in Termen immer als implizit allquantifiziert betrachtet.

Die Unifikation entspricht hingegen eher einem "Erfüllbarkeitsproblem". Im allgemeinen wird hier untersucht, ob es zu einer Algebra A (bzw. einer Menge von Algebren) eine Substitution σ gibt, so dass $A \models s\sigma \equiv t\sigma$ gilt. Je nachdem, welche Algebra A bzw. welche Menge von Algebren man betrachtet, kann dieses Problem entscheidbar oder unentscheidbar sein und es existieren jeweils unterschiedliche Unifikationsalgorithmen.

Wir wollen im folgenden die einfachste Unifikationsproblematik untersuchen. Hierbei wird überprüft, ob es eine Substitution σ gibt, so dass die Gleichung $s\sigma \equiv t\sigma$ in *allen* Algebren gilt. Mit anderen Worten, man untersucht, ob man durch eine geeignete Substitution σ die beiden Terme s und t *syntaktisch* gleich machen kann. Beispielsweise ist das Unifikationsproblem $\mathsf{g}(\mathsf{f}(x), y) =^? \mathsf{g}(y, \mathsf{f}(z))$ lösbar und hat unter anderem die Lösung $\sigma = \{x/z, y/\mathsf{f}(z)\}$. Solche Lösungssubstitutionen bezeichnet man als *Unifikatoren*. Die folgende Definition führt die benötigten Begriffe ein, wobei wir im folgenden nicht nur einzelne Gleichungen, sondern Systeme von Gleichungen als Unifikationsprobleme betrachten.

Definition 5.1.1 (Unifikation) *Zwei Terme s und t sind* unifizierbar*, falls es eine Substitution σ mit $s\sigma = t\sigma$ gibt. Solch eine Substitution heißt ein* Unifikator *von s und t.*

Ein Unifikationsproblem *über Σ und $\mathcal{V}$ ist eine endliche Menge von Termgleichungen $S = \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$ mit $s_i, t_i \in \mathcal{T}(\Sigma, \mathcal{V})$. Eine Substitution σ ist eine* Lösung *oder ein* Unifikator *des Unifikationsproblems S gdw. $s_i\sigma = t_i\sigma$ für alle $1 \leq i \leq n$ gilt. Die Menge aller Unifikatoren von S bezeichnen wir mit $U(S)$.*

Wir benötigen die Unifikation, um die Konfluenz von TESen zu überprüfen und um TESe zu vervollständigen. Unifikation spielt aber auch in anderen Bereichen der Informatik eine wichtige Rolle. Beispielsweise ist Unifikation der Grundmechanismus bei der Auswertung von logischen Programmen (z.B. in der Sprache PROLOG), vgl. [Gie06]. Außerdem wird Unifikation im Bereich des automatischen Beweisens (z.B. für den sogenannten Resolutionskalkül) benötigt. Weiterhin wird Unifikation bei der Implementierung von polymorph getypten Programmiersprachen zur Typinferenz und -überprüfung verwendet, vgl. [Gie05]. Dies ist beispielsweise bei den meisten modernen funktionalen Sprachen wie HASKELL oder ML der Fall.

Die oben skizzierte Frage der (syntaktischen) Unifikation ist entscheidbar und die Menge aller Unifikatoren zu einem Unifikationsproblem lässt sich auf endliche Weise charakterisieren. Im allgemeinen kann ein Unifikationsproblem natürlich mehrere Unifikatoren haben. Das obige Unifikationsproblem $\mathsf{g}(\mathsf{f}(x), y) =^? \mathsf{g}(y, \mathsf{f}(z))$ hat neben der Lösung $\sigma = \{x/z, y/\mathsf{f}(z)\}$ z.B. auch die Lösungen $\sigma_1 = \{x/\mathsf{a}, y/\mathsf{f}(\mathsf{a}), z/\mathsf{a}\}$, $\sigma_2 = \{x/\mathsf{f}(z), y/\mathsf{f}(\mathsf{f}(z)), z/\mathsf{f}(z)\}$ und $\sigma_3 = \{z/x, y/\mathsf{f}(x)\}$. Die Substitutionen σ und σ_3 sind insofern allgemeiner als die anderen beiden Substitutionen, da hier nur solche Variablen ersetzt werden, die unbedingt ersetzt werden müssen. Man nennt eine Substitution σ *allgemeiner* als eine andere Substitution σ_1, wenn σ_1 eine Spezialisierung von σ ist, d.h., falls man den Effekt von σ_1 erhält, indem man erst σ anwendet und anschließend die Variablen möglicherweise weiter instantiiert. In der Tat gilt $\sigma_1 = \sigma \circ \delta_1$, wobei $\delta_1 = \{z/\mathsf{a}\}$ ist. Ebenso gilt $\sigma_2 = \sigma \circ \delta_2$ mit $\delta_2 = \{z/\mathsf{f}(z)\}$ und $\sigma_3 = \sigma \circ \delta_3$ mit $\delta_3 = \{z/x\}$. Die Substitution σ ist ein *allgemeinster* Uni-

fikator von $\mathsf{g}(\mathsf{f}(x), y)$ und $\mathsf{g}(y, \mathsf{f}(z))$. Wir lassen im Folgenden das Kompositionszeichen "$\circ$" oft weg. Dann bezeichnet "$\sigma\delta$" die Substitution "$\sigma \circ \delta$" und man hat $t\sigma\delta = (t\sigma)\delta = t(\sigma \circ \delta)$.

Definition 5.1.2 (Allgemeinster Unifikator) *Eine Substitution σ ist* allgemeiner *als eine Substitution σ' gdw. es eine Substitution δ mit $\sigma' = \sigma \circ \delta$ gibt. Eine Substitution σ ist* allgemeinster Unifikator *(engl.* most general unifier, mgu*) eines Unifikationsproblems S gdw. $\sigma \in U(S)$ ist und σ allgemeiner als alle $\sigma' \in U(S)$ ist.*

In der Tat ist im obigen Beispiel auch σ_3 ein allgemeinster Unifikator von $\mathsf{g}(\mathsf{f}(x), y)$ und $\mathsf{g}(y, \mathsf{f}(z))$. Der allgemeinste Unifikator eines Unifikationsproblems ist aber bis auf Variablenumbenennungen eindeutig.

Lemma 5.1.3 (Eindeutigkeit allgemeinster Substitutionen) *Seien σ und σ' zwei Substitutionen. Falls σ allgemeiner als σ' und σ' allgemeiner als σ ist, so existiert eine Variablenumbenennung δ mit $\sigma' = \sigma\delta$. Hierbei wird eine Substitution δ als Variablenumbenennung bezeichnet, falls δ injektiv ist und $x\delta \in \mathcal{V}$ für alle $x \in \mathcal{V}$ gilt.*

Beweis. Da σ allgemeiner als σ' und umgekehrt ist, existieren Substitutionen ρ und ρ' mit $\sigma' = \sigma\rho$ und $\sigma = \sigma'\rho'$. Hierbei kann man offensichtlich ρ so wählen, dass $DOM(\rho) \subseteq VRAN(\sigma)$ ist. Hierbei bezeichnet $VRAN(\sigma)$ die Variablen im *Range* von σ, d.h., $VRAN(\sigma) = \mathcal{V}(\{y\sigma | y \in \mathcal{V}\})$. (Üblicherweise wird beim Range nur $y\sigma$ für alle $y \in DOM(\sigma)$ betrachtet; wir benötigen hier jedoch $y\sigma$ für alle $y \in \mathcal{V}$.)

Man erhält $\sigma = \sigma'\rho' = \sigma\rho\rho'$ und damit $x = x\rho\rho'$ für alle $x \in VRAN(\sigma)$. Damit gilt:

$$x\rho \in \mathcal{V} \text{ für alle } x \in \mathcal{V} \tag{5.1}$$

Hierzu betrachten wir die beiden Fälle $x \in DOM(\rho)$ und $x \notin DOM(\rho)$. Für $x \in DOM(\rho)$ gilt $x \in VRAN(\sigma)$. Wäre $x\rho \notin \mathcal{V}$, so wäre auch $x\rho\rho' \notin \mathcal{V}$ und somit $x\rho\rho' \neq x$. Für $x \notin DOM(\rho)$ gilt $x\rho = x \in \mathcal{V}$.

Außerdem ist ρ injektiv auf $VRAN(\sigma)$. Es gilt also

$$\text{Aus } u\rho = v\rho \text{ folgt } u = v \text{ für alle } u, v \in VRAN(\sigma). \tag{5.2}$$

Der Grund ist, dass aus $u\rho = v\rho$ dann auch $u\rho\rho' = v\rho\rho'$ folgt. Damit ergibt sich $u = u\rho\rho' = v\rho\rho' = v$.

Allerdings ist ρ nicht unbedingt auf ganz $\mathcal{V}$ injektiv und somit noch nicht unbedingt eine Variablenumbenennung. Die Substitution ρ hat die Gestalt

$$\rho = \{x_1/y_1, \ldots, x_n/y_n\}$$

wobei alle y_i Variablen sind (nach (5.1)) und aufgrund der Injektivität (5.2) sind alle y_i paarweise verschieden. Außerdem gilt $DOM(\rho) = \{x_1, \ldots, x_n\} \subseteq VRAN(\sigma)$.

Die Verletzung der Injektivität liegt daran, dass es Variablen y_i geben kann, die nicht in $\{x_1, \ldots, x_n\}$ auftreten. Dann gilt nämlich $x_i\rho = y_i$ und $y_i\rho = y_i$ und somit ist ρ nicht injektiv. Aus diesem Grund müssen wir ρ so zu einer Substitution δ erweitern (bzw. verändern), dass auch solche Variablen y_i in den Domain mit aufgenommen werden. Seien $y_{i_1}, \ldots, y_{i_k}$ die Variablen aus $\{y_1, \ldots, y_n\}$, die nicht im Domain $\{x_1, \ldots, x_n\}$ von ρ auftreten. Hierbei seien die y_{i_l} paarweise verschieden, d.h., es gibt k solche Variablen y_i.

Das bedeutet, dass auch k Variablen aus $\{x_1, \ldots, x_n\}$ nicht in $\{y_1, \ldots, y_n\}$ auftreten. Diese Variablen seien $x_{j_1}, \ldots, x_{j_k}$. Um ρ zu eine Variablenumbenennung δ zu erweitern, fügen wir nun die Variablen $y_{i_1}, \ldots, y_{i_k}$ dem Domain hinzu. Damit die entstehende Substitution injektiv wird, müssen wir diese Variablen auf solche Variablen abbilden, auf die bislang keine Variable aus dem Domain abgebildet wird. Daher werden die neuen Variablen $y_{i_1}, \ldots, y_{i_k}$ auf die Variablen $x_{j_1}, \ldots, x_{j_k}$ abgebildet. Die genaue Zuordnung von y_{i_l} zu x_{j_d} ist unerheblich, solange jedes x_{j_d} genau einem y_{i_l} zugeordnet wird. Wir definieren daher z.B.

$$\delta = \rho \cup \{y_{i_1}/x_{j_1}, \ldots, y_{i_k}/x_{j_k}\} = \{x_1/y_1, \ldots, x_n/y_n, y_{i_1}/x_{j_1}, \ldots, y_{i_k}/x_{j_k}\}.$$

Um diese Konstruktion an einem Beispiel zu illustrieren, sei $\sigma = \{v/u, w/v\}$ und $\sigma' = \{u/v\}$. Es existieren Substitutionen ρ und ρ' mit $\sigma' = \sigma\rho$ und $\sigma = \sigma'\rho'$, nämlich $\rho = \{u/v, v/w\}$ und $\rho' = \{v/u, w/v\}$. Die Substitutionen ρ und ρ' sind aber nicht injektiv und daher keine Variablenumbenennungen. Beispielsweise gilt $v\rho = w = w\rho$ und $v \neq w$. Für die Substitution ρ existiert eine Variable y_{i_1} (nämlich w), die zwar für eine andere Variable substituiert wird, aber selbst nicht in $DOM(\rho)$ auftritt. Ebenso existiert daher auch eine Variable x_{j_1} (nämlich u), die bisher nicht auf der rechten Seite eines Substitutionspaars von ρ auftritt. Wir bilden daher aus ρ die Variablenumbenennung δ, indem wir ρ um das Substitutionspaar w/u ergänzen. So ergibt sich die Variablenumbenennung $\delta = \{u/v, v/w, w/u\}$, für die weiterhin $\sigma' = \sigma\delta$ gilt.

In unserem Beweis müssen wir nun die folgenden Aussagen im allgemeinen Fall zeigen:

$$\sigma' = \sigma\delta \tag{5.3}$$

$$x\delta \in \mathcal{V} \text{ für alle } x \in \mathcal{V} \tag{5.4}$$

$$\text{Aus } u\delta = v\delta \text{ folgt } u = v \text{ für alle } u, v \in \mathcal{V}. \tag{5.5}$$

Die Aussage (5.3) ergibt sich, da für alle $x \in VRAN(\sigma)$ immer $x\delta = x\rho$ gilt. Der Grund ist, dass sich δ von ρ nur auf den Variablen $y_{i_1}, \ldots, y_{i_k}$ unterscheidet. Es gilt jedoch $y_{i_l} \notin VRAN(\sigma)$, denn ansonsten hätte man $x_{i_l}\rho = y_{i_l}$ und $y_{i_l}\rho = y_{i_l}$ im Widerspruch zur Injektivität von ρ auf $VRAN(\sigma)$, vgl. (5.2).

Die Aussage (5.4) ergibt sich, da alle x_i und alle y_i Variablen sind (nach (5.1)). Die Injektivität (Aussage 5.5) zeigt man wie folgt: Falls $u, v \in DOM(\delta)$ sind, so folgt die Aussage daraus, dass die rechten Seiten der Substitutionspaare (d.h., die Variablen $y_1, \ldots, y_n, x_{j_1}, \ldots, x_{j_k}$) paarweise verschieden sind. Falls $u, v \notin DOM(\delta)$ sind, so folgt die Aussage trivialerweise. Falls nun $u \in DOM(\delta)$ und $v \notin DOM(\delta)$ sind, so gilt nach der Definition von δ auch $u\delta \in DOM(\delta)$ und $v\delta = v \notin DOM(\delta)$. Dann können aber $u\delta$ und $v\delta$ nicht gleich sein. Der Fall $u \notin DOM(\delta)$ und $v \in DOM(\delta)$ ist analog. □

Im Beispiel gibt es tatsächlich eine Variablenumbenennung δ mit $\sigma = \sigma_3\delta$ und $\sigma_3 = \sigma\delta$, nämlich $\delta = \{x/z, z/x\}$.

Es wird sich herausstellen, dass jedes lösbare Unifikationsproblem auch allgemeinste Unifikatoren besitzt. Mit einem allgemeinsten Unifikator σ kann man dann die (potentiell unendliche) Menge der Lösungen des Unifikationsproblems endlich repräsentieren. Die Unifikatoren sind gerade alle Spezialisierungen von σ, d.h., σ und alle Substitutionen, die weniger allgemein als σ sind.

Lemma 5.1.4 (Darstellung der Lösungen eines Unifikationsproblems) *Sei S ein Unifikationsproblem über Σ und $\mathcal{V}$ und sei σ ein allgemeinster Unifikator von S. Dann gilt $U(S) = \{\sigma\delta \mid \delta \in SUB(\Sigma, \mathcal{V})\}$.*

Beweis. Da σ Unifikator ist, gilt für alle $s =^? t \in S$ jeweils $s\sigma = t\sigma$ und somit auch $s\sigma\delta = t\sigma\delta$. Andererseits existiert für jeden Unifikator σ' von S eine Substitution δ mit $\sigma' = \sigma\delta$. □

Wir führen nun ein Verfahren ein, das zu einem Unifikationsproblem entscheidet, ob es lösbar ist und gegebenenfalls einen allgemeinsten Unifikator berechnet. Die Idee besteht darin, das Unifikationsproblem so zu transformieren, bis es in einer "gelösten Form" ist, aus der man die Lösung unmittelbar ablesen kann.

Definition 5.1.5 (Gelöste Form eines Unifikationsproblems) *Ein Unifikationsproblem $S = \{x_1 =^? t_1, \ldots, x_n =^? t_n\}$ ist in* gelöster Form *gdw. die x_i paarweise verschiedene Variablen sind, die in den Termen $t_1, \ldots, t_n$ nicht vorkommen. Zu einem Unifikationsproblem in gelöster Form wie oben definieren wir die Substitution $\sigma_S = \{x_1/t_1, \ldots, x_n/t_n\}$.*

Der Vorteil von Unifikationsproblemen S in gelöster Form ist, dass die entsprechende Substitution σ_S ein allgemeinster Unifikator ist.

Lemma 5.1.6 (Lösung eines Unifikationsproblems in gelöster Form) *Sei S ein Unifikationsproblem in gelöster Form. Dann ist σ_S ein allgemeinster Unifikator von S, wobei für alle Unifikatoren σ' von S jeweils $\sigma' = \sigma_S\sigma'$ gilt.*

Beweis. Da die x_i nicht in den t_i auftreten, folgt $x_i\sigma_S = t_i = t_i\sigma_S$, d.h., σ_S ist ein Unifikator von S. Sei nun σ' ein weiterer Unifikator von S. Zu zeigen ist, dass dann $\sigma' = \sigma_S\sigma'$ gilt: Für x_i gilt $x_i\sigma' = t_i\sigma' = x_i\sigma_S\sigma'$. Für sonstige Variablen x gilt $x\sigma' = x\sigma_S\sigma'$. □

Zur Lösung von Unifikationsproblemen geben wir nun ein Verfahren an, um ein Unifikationsproblem in gelöste Form zu überführen. Hierzu führen wir die folgende Transformation von Unifikationsproblemen ein:

Definition 5.1.7 (Transformation von Unifikationsproblemen) *Wir definieren die folgende Relation $\Longrightarrow$ auf Unifikationsproblemen durch vier Transformationsregeln.*

Löschen	$S \uplus \{t =^? t\}$	$\Longrightarrow$	S
Termreduktion	$S \uplus \{f(s_1, .., s_n) =^? f(t_1, .., t_n)\}$	$\Longrightarrow$	$S \cup \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$
Vertauschen	$S \uplus \{t =^? x\}$	$\Longrightarrow$	$S \cup \{x =^? t\}$, *falls* $t \notin \mathcal{V}$
Variablenreduktion	$S \uplus \{x =^? t\}$	$\Longrightarrow$	$\{u\sigma =^? v\sigma \mid u =^? v \in S\} \cup \{x =^? t\}$, *falls* $\sigma = \{x/t\}, x \notin \mathcal{V}(t), x \in \mathcal{V}(S)$

Hierbei bezeichnet $\uplus$ die disjunkte Vereinigung, so dass die bearbeitete Gleichung aus dem aktuellen Unifikationsproblem (gegebenenfalls) entfernt wird. Die erste Regel löscht triviale Gleichungen, die nicht zum Unifikationsproblem beitragen. Die zweite Regel ersetzt Gleichungen zwischen Termen, die mit demselben Funktionssymbol gebildet werden, durch die Gleichungen zwischen ihren Argumenten. Die dritte Regel orientiert Gleichungen so,

dass Variablen auf der linken Seite von Gleichungen auftreten. Die Bedingung $t \notin \mathcal{V}$ ist hierbei nötig, um eine unendliche Anwendung dieser Regel zu verhindern.

Die letzte Regel verwendet Gleichungen $x =^? t$, die bereits in gelöster Form sind, um diese Teillösung auf den Rest des Unifikationsproblems anzuwenden. Dies ist natürlich nur sinnvoll, wenn die Variable x auch noch im restlichen Unifikationsproblem auftritt (ansonsten ließe sich die Regel unendlich oft anwenden). Die Bedingung $x \notin \mathcal{V}(t)$ bezeichnet man als *Occur Check*, da sie überprüft, ob die Variable x in dem mit ihr zu unifizierenden Term t vorkommt. Falls dies der Fall wäre und $t \neq x$ ist, so wäre das Unifikationsproblem nicht lösbar und man spricht von einem *Occur Failure*. Der Grund ist, dass bei jeder Substitution σ der Term $x\sigma$ immer ein echter Teilterm von $t\sigma$ ist. Ein Beispiel ist das (unlösbare) Unifikationsproblem $x = \mathsf{f}(x)$.

Es gibt insgesamt zwei Gründe, warum ein Unifikationsproblem unlösbar sein kann. Neben dem Occur Failure gibt es den *Clash Failure*, der entsteht, wenn zwei Terme mit verschiedenem führendem Funktionssymbol miteinander unifiziert werden sollen. Beispielsweise sind $\mathsf{f}(x)$ und $\mathsf{g}(y, z)$ offensichtlich nicht miteinander unifizierbar.

Beispiel 5.1.8 Das folgende Beispiel illustriert die Arbeitsweise der obigen Regeln, um das Unifikationsproblem $\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{g}(x, x)) =^? \mathsf{g}(x, \mathsf{g}(x, y))$ zu lösen:

$$
\begin{array}{ll}
\{\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{g}(x, x)) =^? \mathsf{g}(x, \mathsf{g}(x, y))\} & \Longrightarrow_{\textbf{Termreduktion}} \\
\{\mathsf{f}(\mathsf{a}) =^? x, \mathsf{g}(x, x) =^? \mathsf{g}(x, y)\} & \Longrightarrow_{\textbf{Vertauschen}} \\
\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{g}(x, x) =^? \mathsf{g}(x, y)\} & \Longrightarrow_{\textbf{Variablenreduktion}} \\
\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a})) =^? \mathsf{g}(\mathsf{f}(\mathsf{a}), y)\} & \Longrightarrow_{\textbf{Termreduktion}} \\
\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}) =^? \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}) =^? y\} & \Longrightarrow_{\textbf{Löschen}} \\
\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}) =^? y\} & \Longrightarrow_{\textbf{Vertauschen}} \\
\{x =^? \mathsf{f}(\mathsf{a}), y =^? \mathsf{f}(\mathsf{a})\} &
\end{array}
$$

Man erkennt, dass auf diese Weise das Unifikationsproblem in gelöste Form überführt wurde.

Mit Hilfe der Transformationsregeln aus Def. 5.1.7 erhält man direkt einen Unifikationsalgorithmus, vgl. [Rob65, MM82]. Hierzu wendet man auf das zu lösende Unifikationsproblem S beliebig Transformationsregeln an, solange dies möglich ist. Hierbei gibt es natürlich Indeterminismen, da man z.B. im obigen Beispiel im dritten Schritt auch "Termreduktion" vor der "Variablenreduktion" hätte anwenden können. Sofern man zum Schluss ein Unifikationsproblem S' in gelöster Form erhält, so liest man daraus den Unifikator $\sigma_{S'}$ ab, der dann auch ein allgemeinster Unifikator für das ursprüngliche Problem ist. Ist das Resultat nicht in gelöster Form, so ist das Unifikationsproblem nicht lösbar.

Algorithmus $\mathsf{UNIFY}(S)$
Eingabe: Ein Unifikationsproblem S.
Ausgabe: Ein allgemeinster Unifikator von S, sofern S lösbar ist und sonst "*False*".

1. Solange es ein S' mit $S \Longrightarrow S'$ gibt, setze $S := S'$ und gehe zu 1.
2. Falls S in gelöster Form ist, dann gib σ_S aus.
   Sonst gib "*False*" aus.

Der folgende Satz zeigt die Korrektheit des Verfahrens.

Satz 5.1.9 (Korrektheit des Unifikationsalgorithmus)
Sei S ein Unifikationsproblem.

(a) Die Relation $\Longrightarrow$ ist fundiert.

(b) Falls $S \Longrightarrow S'$, dann gilt $U(S) = U(S')$.

(c) Falls S lösbar und in Normalform bezüglich $\Longrightarrow$ ist, dann ist S in gelöster Form.

(d) Der Algorithmus UNIFY *terminiert und ist korrekt.*

Beweis.

(a) Der Fundiertheitsbeweis ist nicht trivial, da durch die "Variablenreduktion" die Anzahl der Zeichen eines Unifikationsproblems vergrößert werden kann, wie in Bsp. 5.1.8 deutlich wird.

Wir sagen, eine Variable x ist *gelöst* gdw. sie genau einmal in S auftritt, und zwar auf der linken Seite einer Gleichung $x =^? t$, wobei $x \notin \mathcal{V}(t)$. Wir zeigen nun die Fundiertheit von $\Longrightarrow$, indem wir eine Funktion angeben, die jedes Unifikationsproblem S auf einen Tripel (n_1, n_2, n_3) von natürlichen Zahlen abbildet.

- n_1 ist die Anzahl der Variablen in S, die nicht gelöst sind
- n_2 ist die Anzahl der Zeichen in S, d.h. $\sum_{(s=^?t)\in S} |s| + |t|$
- n_3 ist die Anzahl von Gleichungen der Form $t =^? x$ in S, wobei $t \notin \mathcal{V}$.

Zur Illustration betrachten wir noch einmal das Unifikationsproblem aus Bsp. 5.1.8 und untersuchen, wie sich der Tripel (n_1, n_2, n_3) hier verändert.

$\{\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{g}(x,x)) =^? \mathsf{g}(x, \mathsf{g}(x,y))\}$	$\Longrightarrow_{\textbf{Termreduktion}}$	$(2, 11, 0)$
$\{\mathsf{f}(\mathsf{a}) =^? x, \mathsf{g}(x,x) =^? \mathsf{g}(x,y)\}$	$\Longrightarrow_{\textbf{Vertauschen}}$	$(2, 9, 1)$
$\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{g}(x,x) =^? \mathsf{g}(x,y)\}$	$\Longrightarrow_{\textbf{Variablenreduktion}}$	$(2, 9, 0)$
$\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a})) =^? \mathsf{g}(\mathsf{f}(\mathsf{a}), y)\}$	$\Longrightarrow_{\textbf{Termreduktion}}$	$(1, 12, 0)$
$\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}) =^? \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}) =^? y\}$	$\Longrightarrow_{\textbf{Löschen}}$	$(1, 10, 1)$
$\{x =^? \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{a}) =^? y\}$	$\Longrightarrow_{\textbf{Vertauschen}}$	$(1, 6, 1)$
$\{x =^? \mathsf{f}(\mathsf{a}), y =^? \mathsf{f}(\mathsf{a})\}$		$(0, 6, 0)$

Man erkennt nun, dass aus $S \Longrightarrow S'$ folgt, dass der Tripel, der zu S gehört, größer als der Tripel zu S' bezüglich der Relation $(>_{\mathbb{N}})^3_{lex}$ ist. Die folgende Tabelle gibt an, welche Zahlen sich bei den verschiedenen Regeln auf welche Art und Weise verkleinern:

	n_1	n_2	n_3
Löschen	$\geq$	$>$	$=$
Termreduktion	$\geq$	$>$	
Vertauschen	$\geq$	$=$	$>$
Variablenreduktion	$>$		

(b) Wir zeigen $U(S) = U(S')$ falls S' aus S mit Hilfe der Regel "Variablenreduktion" entsteht, da die Aussage für alle anderen Regeln offensichtlich ist. Da $\{x =^? t\}$ in gelöster Form ist, ist $\sigma = \{x/t\}$ nach Lemma 5.1.6 allgemeinster Unifikator dieses Unifikationsproblems. Für alle Substitutionen θ mit $x\theta = t\theta$ gilt nach dem Lemma demnach $\theta = \sigma\theta$. Damit erhält man:

$$\begin{aligned}
\theta \in U(S \uplus \{x =^? t\}) \quad & \text{gdw.} \quad x\theta = t\theta \text{ und } \theta \in U(S) \\
& \text{gdw.} \quad x\theta = t\theta \text{ und } \sigma\theta \in U(S) \\
& \text{gdw.} \quad x\theta = t\theta \text{ und } u\sigma\theta = v\sigma\theta \text{ für alle } u =^? v \in S \\
& \text{gdw.} \quad x\theta = t\theta \text{ und } \theta \in U(\{u\sigma =^? v\sigma \mid u =^? v \in S\}) \\
& \text{gdw.} \quad \theta \in U(\{u\sigma =^? v\sigma \mid u =^? v \in S\} \cup \{x =^? t\})
\end{aligned}$$

(c) Wenn S lösbar ist, kann es keine Gleichungen $f(\ldots) =^? g(\ldots)$ mit $f \neq g$ enthalten, da diese Gleichungen dem Clash Failure entsprechen würden. S kann auch keine Gleichungen $f(\ldots) =^? f(\ldots)$ enthalten, da sonst die "Termreduktion"-Regel anwendbar wäre. Aus $s =^? t \in S$ folgt daher $s \in \mathcal{V}$ oder $t \in \mathcal{V}$. Falls $s \notin \mathcal{V}$, so wäre die "Vertauschen"-Regel anwendbar. Somit haben alle Gleichungen in S die Form $x =^? t$. Hierbei gilt $x \notin \mathcal{V}(t)$. Der Grund ist, dass $t = x$ nicht möglich ist, da sonst die "Löschen"-Regel anwendbar wäre und $x \in \mathcal{V}(t)$ bei $t \neq x$ würde dem Occur Failure entsprechen. Die Variable x kommt darüber hinaus in keiner anderen Gleichung aus S vor, da sonst die "Variablenreduktion" anwendbar wäre. Somit ist S in gelöster Form.

(d) Die Terminierung von UNIFY folgt direkt aus Teil (a). Somit berechnet UNIFY(S) also eine Normalform von S, d.h., ein Unifikationsproblem S' mit $S \Longrightarrow^* S'$, das in Normalform bezüglich $\Longrightarrow$ ist. Durch Induktion über die Länge der Transformation kann man mit Hilfe von Teil (b) zeigen, dass $U(S) = U(S')$ ist. Falls S lösbar ist, ist $U(S) = U(S') \neq \varnothing$ und somit ist auch S' lösbar. Nach Teil (c) ist daher S' in gelöster Form und $\sigma_{S'}$ ist nach Lemma 5.1.6 allgemeinster Unifikator von S' und daher auch von S. Falls S nicht lösbar ist, ist $U(S) = U(S') = \varnothing$. Somit kann S' nicht in gelöster Form sein und der Algorithmus gibt "*False*" aus. □

Aus diesem Satz erhält man sofort den bekannten Unifikationssatz, der sich daraus ergibt, dass UNIFY zu jedem lösbaren Unifikationsproblem einen allgemeinsten Unifikator berechnet.

Korollar 5.1.10 (Unifikationssatz) *Wenn ein Unifikationsproblem lösbar ist, dann hat es einen allgemeinsten Unifikator.*

Die Effizienz des Algorithmus UNIFY lässt sich natürlich noch verbessern, indem man sofort mit "*False*" abbricht, sobald ein Occur oder Clash Failure bemerkt wird. Durch weitere Verbesserungen (insbesondere durch die Repräsentation der Terme als Graphen) lässt sich sogar ein Unifikationsalgorithmus mit linearem Aufwand gewinnen.

Mit Hilfe des Unifikationsalgorithmus UNIFY lässt sich auch sofort ein Matchingalgorithmus konstruieren. (Aus Effizienzgründen wird Matching allerdings üblicherweise getrennt von der Unifikation implementiert.) Solch ein Algorithmus wird natürlich benötigt, um Termersetzung (oder die Auswertung in funktionalen Programmen) automatisch durchzuführen. Ein Term s matcht einen Term t, falls es eine Substitution σ mit $s\sigma = t$ gibt. Im Unterschied zur Unifikation werden hierbei also nur die Variablen der Terme auf linken Seiten von Gleichungen instantiiert. Auf diesen Variablen ist der Matcher daher eindeutig. Das Matching-Problem lässt sich daher leicht auf das Unifikationsproblem reduzieren, indem man die Variablen in t einfach als Konstanten betrachtet. Dies lässt sich zum Beispiel erreichen, indem man alle Variablen x in t zuerst durch neue Konstanten c_x ersetzt, dann s und die Modifikation von t unifiziert und hinterher die Konstanten im berechneten Unifikator wieder durch die ursprünglichen Variablen ersetzt.

Algorithmus $\mathsf{MATCH}(s, t)$
Eingabe: Zwei Terme s und t.
Ausgabe: Ein Matcher σ mit $s\sigma = t$, sofern ein solcher existiert und sonst "*False*".

1. Sei $\theta = \{x/c_x \mid x \in \mathcal{V}(t)\}$, wobei c_x neue Konstanten sind.
2. Berechne $\mathsf{UNIFY}(\{s =^? t\theta\})$.
3. Falls UNIFY das Resultat "*False*" liefert, so gib "*False*" aus und brich ab.
4. Falls UNIFY das Resultat $\{x_1/t_1, \ldots, x_n/t_n\}$ liefert, so gib $\{x_1/t'_1, \ldots, x_n/t'_n\}$ aus. Hierbei entsteht t'_i aus t_i, indem alle in Schritt 1 eingeführten Konstanten c_x durch die jeweilige Variable x ersetzt werden.

Das folgende Beispiel zeigt die Arbeitsweise des Matching-Algorithmus.

Beispiel 5.1.11 Um den Matcher von $\mathsf{g}(x, y)$ und $\mathsf{g}(\mathsf{f}(x), x)$ zu finden, wird zunächst die Substitution $\theta = \{x/c_x\}$ auf $\mathsf{g}(\mathsf{f}(x), x)$ angewendet. Anschließend versucht man, die Terme $\mathsf{g}(x, y)$ und $\mathsf{g}(\mathsf{f}(c_x), c_x)$ zu unifizieren. Hierbei ergibt sich der (allgemeinste) Unifikator $\{x/\mathsf{f}(c_x), y/c_x\}$. Der Algorithmus MATCH gibt daher $\{x/\mathsf{f}(x), y/x\}$ zurück.

5.2 Lokale Konfluenz und kritische Paare

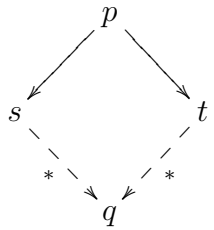
Unser Ziel ist es, für ein Termersetzungssystem die Konfluenz automatisch zu überprüfen. Allerdings ist die Konfluenz eines TES im allgemeinen nicht entscheidbar (siehe z.B. [BN98, Abschnitt 6.1]). Im Folgenden geben wir jedoch ein Kriterium an, mit dem man die Konfluenz dennoch oftmals nachweisen kann. Hierzu schwächen wir die Forderung der Konfluenz etwas ab, indem wir nicht mehr verlangen, dass sich alle Indeterminismen $p \to^* s$ und $p \to^* t$ (nach *beliebig* vielen Schritten) zusammenführen lassen, sondern indem wir nur noch fordern, dass sich Indeterminismen $p \to s$ und $p \to t$ nach *einem* Schritt wieder zusammenführen lassen.

Definition 5.2.1 (Lokale Konfluenz) *Eine Relation $\to$ über einer Menge M heißt* lokal konfluent*, wenn für alle $s, t, p \in M$ gilt:*

Wenn $p \to s$ und $p \to t$,
dann existiert ein $q \in M$ mit $s \to^* q$ und $t \to^* q$.

Ein TES $\mathcal{R}$ heißt lokal konfluent*, wenn die Ersetzungsrelation $\to_{\mathcal{R}}$ lokal konfluent ist.*

Ähnlich zur Konfluenz kann man auch die lokale Konfluenz mit Hilfe eines Diagramms verdeutlichen. Die Bedeutung ist wieder wie folgt: Wenn die durchgezogenen Pfeile existieren, folgt daraus die Existenz der gestrichelten Pfeile und die Existenz eines geeigneten Objekts q.



Trivialerweise ist jede konfluente Relation auch lokal konfluent. Die Umkehrung gilt jedoch nicht:

Beispiel 5.2.2 Sei $\mathcal{R} = \{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{c}, \mathsf{c} \to \mathsf{b}, \mathsf{c} \to \mathsf{d}\}$. Dies lässt sich durch folgendes Bild veranschaulichen:

Man hat den Indeterminismus $\mathsf{b} \to_{\mathcal{R}} \mathsf{a}$ und $\mathsf{b} \to_{\mathcal{R}} \mathsf{c}$, aber diese beiden Terme können wieder zusammengeführt werden, denn es gilt $\mathsf{c} \to^*_{\mathcal{R}} \mathsf{a}$. Ebenso hat man auch den Indeterminismus $\mathsf{c} \to_{\mathcal{R}} \mathsf{b}$ und $\mathsf{c} \to_{\mathcal{R}} \mathsf{d}$, aber auch diese beiden Terme lassen sich durch $\mathsf{b} \to^*_{\mathcal{R}} \mathsf{d}$ wieder zusammenführen. Das TES ist also lokal konfluent. Es ist aber nicht konfluent, denn wir haben $\mathsf{b} \to^*_{\mathcal{R}} \mathsf{a}$ und $\mathsf{b} \to^*_{\mathcal{R}} \mathsf{d}$; die beiden Terme a und d sind aber Normalformen und lassen sich daher nicht mehr zusammenführen.

Beispiel 5.2.2 zeigt, dass die Forderung nach lokaler Konfluenz im allgemeinen zu schwach ist, um die Konfluenz zu garantieren. Fordert man jedoch noch zusätzlich die *Fundiertheit* der Relation $\to$ (bzw. die Terminierung des TES), so ist lokale Konfluenz hinreichend für Konfluenz. Dies ist das fundamentale *Diamond Lemma* von [New42]. (Der Name des Lemmas bezieht sich auf seinen Beweis, in dem eine Art Diamant von Ableitungen konstruiert wird.)

Satz 5.2.3 (Diamond Lemma, Satz von Newman [New42]) *Sei $\to$ eine fundierte Relation über einer Menge M. Dann ist $\to$ konfluent gdw. $\to$ lokal konfluent ist.*

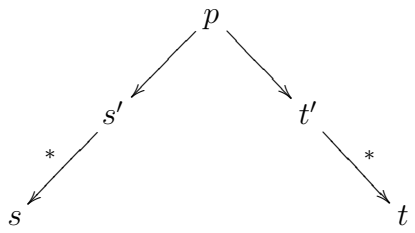
Beweis. Aus der Konfluenz folgt trivialerweise die lokale Konfluenz. Wir zeigen nun die Rückrichtung und setzen hierzu voraus, dass $\to$ lokal konfluent ist. Für alle $p \in M$ muss also folgende Aussage gezeigt werden:

$$\text{Falls } s \leftarrow^* p \to^* t, \text{ dann } s \downarrow t. \tag{5.6}$$

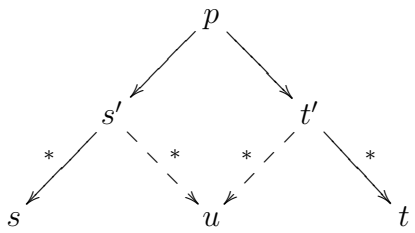
Wie üblich bedeutet "$s \downarrow t$", dass s und t zusammenführbar sind, d.h., dass es ein $q \in M$ mit $s \to^* q \leftarrow^* t$ gibt.

Wir zeigen (5.6) durch Induktion über p. Hierzu benutzen wir noethersche Induktion und verwenden $\to$ als Induktionsrelation. Dies ist möglich, da $\to$ nach Voraussetzung fundiert ist. Wenn wir (5.6) für p zeigen wollen, so dürfen wir es also für alle p' mit $p \to p'$ voraussetzen.

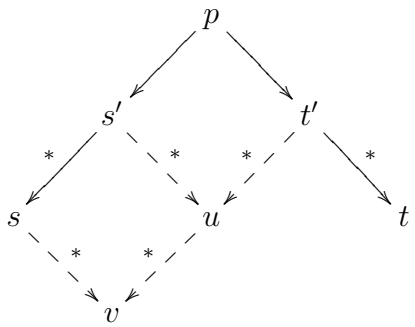
Wir betrachten nun die "kritische Situation" $s \leftarrow^* p \to^* t$. Falls $s = p$ ist, so folgt $s \to^* t$ und damit $s \downarrow t$. Ebenso folgt bei $p = t$, dass $t \to^* s$ und damit $s \downarrow t$ gilt. Ansonsten haben die beiden Reduktionsfolgen $s \leftarrow^* p$ und $p \to^* t$ beide mindestens die Länge 1 und somit existieren s' und t' mit



Aufgrund der lokalen Konfluenz gibt es also ein Objekt u mit $s' \to^* u \leftarrow^* t'$, d.h.,

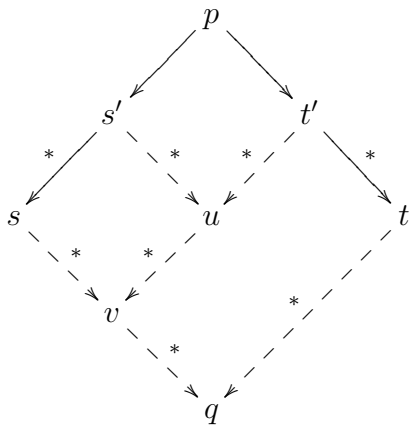


Da $p \to s'$ gilt, ist das Objekt s' "kleiner" in der Induktionsrelation als p. Die Eigenschaft (5.6) darf daher für s' aufgrund der Induktionshypothese vorausgesetzt werden. Mit anderen Worten, wenn man von s' aus in beliebig vielen Schritten zu zwei verschiedenen Objekten (wie s und u) kommt, so gibt es ein Objekt v, zu dem diese beiden Objekte zusammenführbar sind. Man erhält also



Schließlich gilt auch $p \to t'$, d.h., das Objekt t' ist ebenfalls "kleiner" als p in der Induktionsrelation. Da $v \leftarrow^* t' \to^* t$ gilt, folgt daher aus der Induktionshypothese, dass v und t

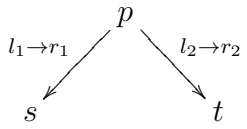
zusammenführbar sind, d.h., es existiert ein Objekt q mit $v \to^* q \leftarrow^* t$.



Damit erhält man also $s \to^* q \leftarrow^* t$, d.h., s und t sind zusammenführbar. Das hierbei entstandene Diagramm ähnelt (entfernt) einem Diamanten; daher der Name des Satzes. □

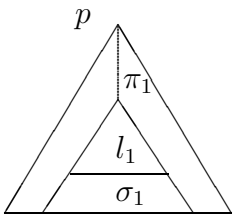
Mit dem Diamond Lemma kann der Konfluenztest "lokalisiert" werden. Man muss also nur bei Verzweigungen nach *einem* Schritt zeigen, dass sie wieder zusammenführbar sind. Das Diamond Lemma garantiert dann, dass dies auch bei Verzweigungen nach beliebig vielen Schritten möglich ist. Dieses Resultat ist insbesondere dann von Vorteil, wenn die Relation $\to$ endlich darstellbar ist (wie bei (endlichen) Termersetzungssystemen). Wir werden nun zeigen, dass dort die lokale Konfluenz entscheidbar ist. Wenn man also die Terminierung sicher gestellt hat, ist dann sogar die Konfluenz entscheidbar. Im Rahmen unseres generellen Vorgehens (Abbildung 3.1), bei dem ja zunächst die Terminierung überprüft wird, kann also die Konfluenz nun in der Tat bestimmt werden. Es gibt darüber hinaus Kriterien, die auch für die Konfluenz bei nicht-terminierenden Regeln hinreichend sind (dies ist insbesondere bei funktionalen Programmen wichtig). Hierzu sei auf Abschnitt 5.3 verwiesen.

Nach dem Diamond Lemma müssen wir bei terminierenden TESen also lediglich die lokale Konfluenz untersuchen, d.h., wir müssen alle *kritischen Situationen* der Form



überprüfen und feststellen, ob man s und t jeweils zusammenführen kann. Leider gibt es natürlich im allgemeinen unendlich viele Terme und daher auch unendlich viele solche kritischen Situationen. Um die (lokale) Konfluenzüberprüfung automatisch durchführen zu können, müssen wir uns daher auf endlich viele kritische Situationen beschränken. Unser Ziel ist also, eine derartige endliche Menge von kritischen Situationen zu finden, so dass aus der Zusammenführbarkeit von diesen kritischen Situationen bereits die Zusammenführbarkeit *aller* kritischen Situationen und damit die lokale Konfluenz folgt.

Hierzu analysieren wir die verschiedenen Möglichkeiten, an welchen Stellen die Termersetzungsregeln bei einer kritischen Situation angewendet werden. Wir gehen davon aus, dass der Schritt $p \to_{\mathcal{R}} s$ mit Hilfe der Regel $l_1 \to r_1$ an der Stelle π_1 mit der Substitution σ_1 durchgeführt wurde. Der Term p lässt sich daher grafisch wie folgt veranschaulichen.



Analog dazu wurde bei dem Schritt $p \to_{\mathcal{R}} t$ die Regel $l_2 \to r_2$ an der Stelle π_2 mit der Substitution σ_2 verwendet. Wir haben also

- $p|_{\pi_1} = l_1\sigma_1$ und $s = p[r_1\sigma_1]_{\pi_1}$
- $p|_{\pi_2} = l_2\sigma_2$ und $t = p[r_2\sigma_2]_{\pi_2}$

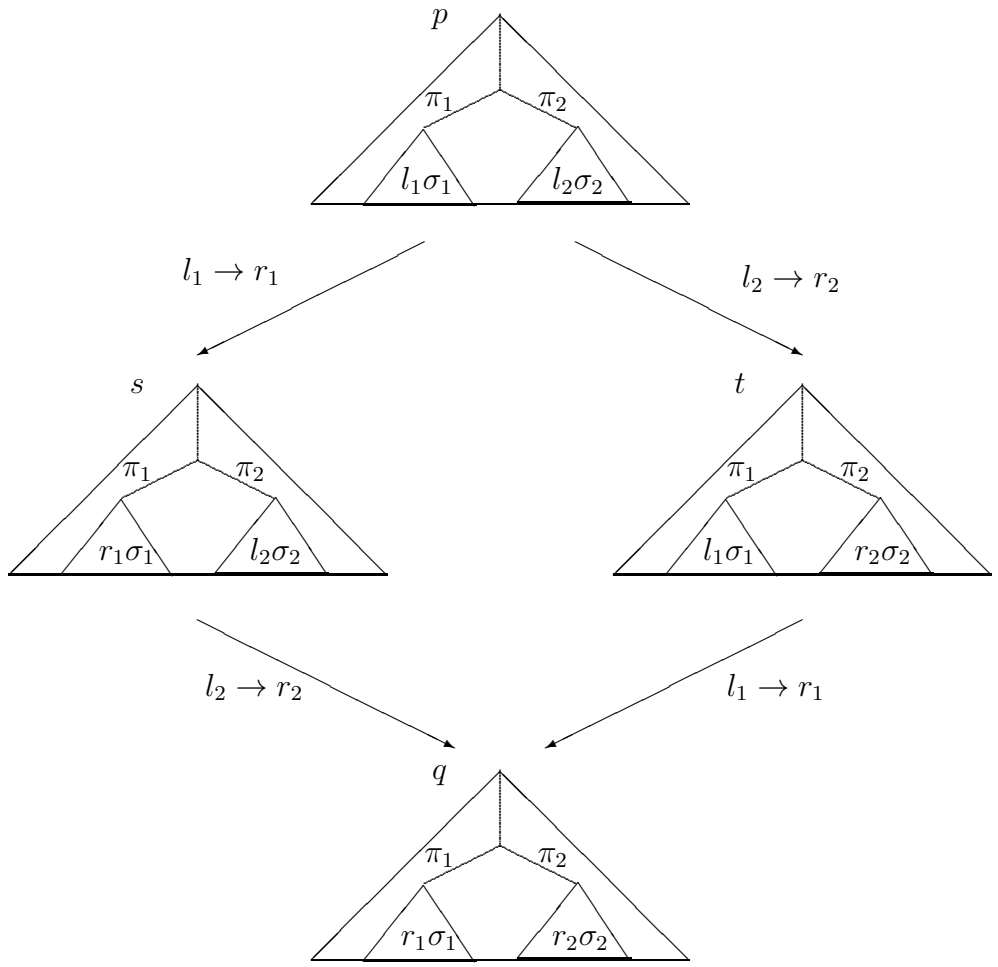
Wir unterscheiden nun die verschiedenen Möglichkeiten, an welcher Stelle sich die Positionen π_1 und π_2 in dem Term p befinden können. Wir wollen also analysieren, wie die beiden Redexe $l_1\sigma_1$ und $l_2\sigma_2$ miteinander interagieren bzw. ob die Reduktion des einen Redex den anderen Redex zerstören kann. Nur in diesem Fall kann dies zu einer Verletzung der Konfluenz führen.

Fall 1: Die Stellen π_1 und π_2 sind voneinander unabhängig, d.h., $\pi_1 \perp \pi_2$

In diesem Fall liegen die beiden Stellen nicht übereinander. Es gilt also weder $\pi_2 \geq_{\mathbb{N}^*} \pi_1$ noch $\pi_1 \geq_{\mathbb{N}^*} \pi_2$. In diesem Fall ist die Reduktion von p zu s bzw. zu t unkritisch, denn man hat

- $p = p[l_1\sigma_1]_{\pi_1}[l_2\sigma_2]_{\pi_2}$
- $s = p[r_1\sigma_1]_{\pi_1}[l_2\sigma_2]_{\pi_2}$
- $t = p[l_1\sigma_1]_{\pi_1}[r_2\sigma_2]_{\pi_2}$
- $s \to_{\mathcal{R}} q$ und $t \to_{\mathcal{R}} q$ mit $q = p[r_1\sigma_1]_{\pi_1}[r_2\sigma_2]_{\pi_2}$

Somit lassen sich s und t in einem Schritt zusammenführen. Grafisch lässt sich diese Situation folgendermaßen verdeutlichen:

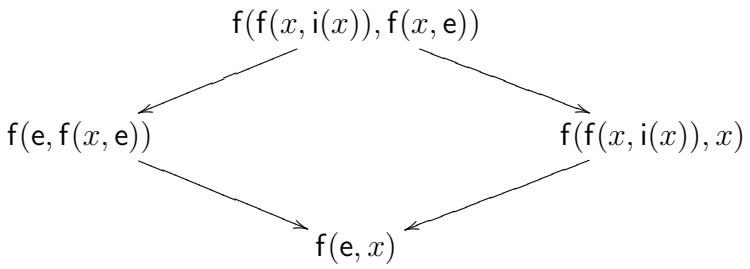


Man erkennt also, dass solche Situationen nicht betrachtet werden müssen, um die lokale Konfluenz nachzuweisen, da hier die entstehenden Terme immer zusammenführbar sind.

Als Beispiel betrachten wir wieder unser Gruppen-TES.

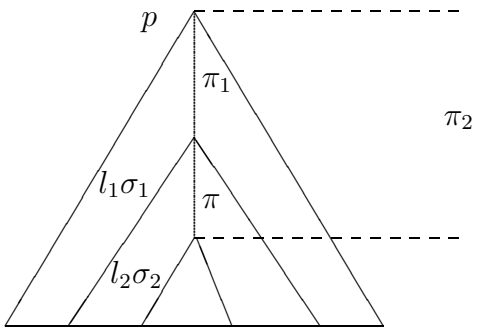
$$\begin{aligned} \mathsf{f}(x, \mathsf{f}(y, z)) &\to \mathsf{f}(\mathsf{f}(x, y), z) \\ \mathsf{f}(x, \mathsf{e}) &\to x \\ \mathsf{f}(x, \mathsf{i}(x)) &\to \mathsf{e} \end{aligned}$$

Hier haben wir die kritische Situation $\mathsf{f}(\mathsf{e}, \mathsf{f}(x, \mathsf{e})) \leftarrow_{\mathcal{R}} \mathsf{f}(\mathsf{f}(x, \mathsf{i}(x)), \mathsf{f}(x, \mathsf{e})) \to_{\mathcal{R}} \mathsf{f}(\mathsf{f}(x, \mathsf{i}(x)), x)$. Die Reduktionen finden hier an den Positionen 1 und 2 statt. Da $1 \perp 2$ gilt, sind solche Terme aber immer zusammenführbar und es gilt:



Fall 2: π_1 liegt über π_2 ($\pi_2 \geq_{\mathbb{N}^*} \pi_1$)

Der 3. Fall, in dem π_2 über π_1 liegt, ist hierzu analog, so dass wir nur diesen zweiten Fall betrachten. Nun gilt $\pi_2 = \pi_1\pi$ für eine Position π (wobei $\pi = \epsilon$ möglich ist), d.h., p hat folgende Gestalt:



Anstelle der kritischen Situation $s \leftarrow_{\mathcal{R}} p \rightarrow_{\mathcal{R}} t$ betrachten wir nun nur die kritische Situation $s|_{\pi_1} \leftarrow_{\mathcal{R}} p|_{\pi_1} \rightarrow_{\mathcal{R}} t|_{\pi_1}$. Falls sich $s|_{\pi_1}$ und $t|_{\pi_1}$ zusammenführen lassen, so lassen sich (da die Ersetzungsrelation unter Kontexten abgeschlossen ist) auch s und t zusammenführen. Wir haben nämlich

- $p = p[l_1\sigma_1]_{\pi_1} = p[l_1\sigma_1[l_2\sigma_2]_{\pi}]_{\pi_1}$
- $s = p[r_1\sigma_1]_{\pi_1}$
- $t = p[r_2\sigma_2]_{\pi_2} = p[l_1\sigma_1[r_2\sigma_2]_{\pi}]_{\pi_1}$

und daher

- $p|_{\pi_1} = l_1\sigma_1 = l_1\sigma_1[l_2\sigma_2]_{\pi}$
- $s|_{\pi_1} = r_1\sigma_1$
- $t|_{\pi_1} = l_1\sigma_1[r_2\sigma_2]_{\pi}$

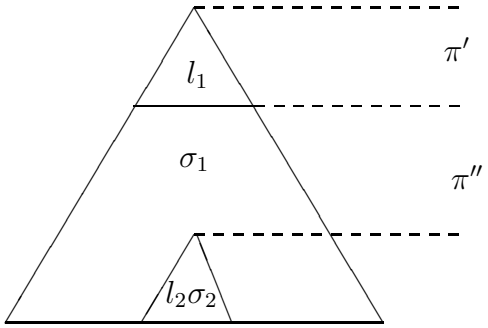
Falls es ein q mit $s|_{\pi_1} \rightarrow^*_{\mathcal{R}} q \leftarrow^*_{\mathcal{R}} t|_{\pi_1}$ gibt, so folgt damit auch $s \rightarrow^*_{\mathcal{R}} p[q]_{\pi_1} \leftarrow^*_{\mathcal{R}} t$ nach Lemma 3.1.13, d.h., s und t sind dann ebenfalls zusammenführbar.

Wir treffen nun eine Fallunterscheidung danach, wie weit die beiden Redexe $l_1\sigma_1$ und $l_2\sigma_2$ voneinander entfernt sind. Wir wissen ja, dass $l_1\sigma_1|_\pi = l_2\sigma_2$ ist. Die Frage ist nun, wie tief die Stelle π in $l_1\sigma_1$ liegt, d.h., ob π eine Stelle von l_1 ist, an der keine Variable steht, oder ob π im "Substitutionsteil" liegt.

Fall 2.1: $l_2\sigma_2$ liegt im Substitutionsteil, d.h., $\pi \notin Occ(l_1)$ oder $l_1|_\pi \in \mathcal{V}$

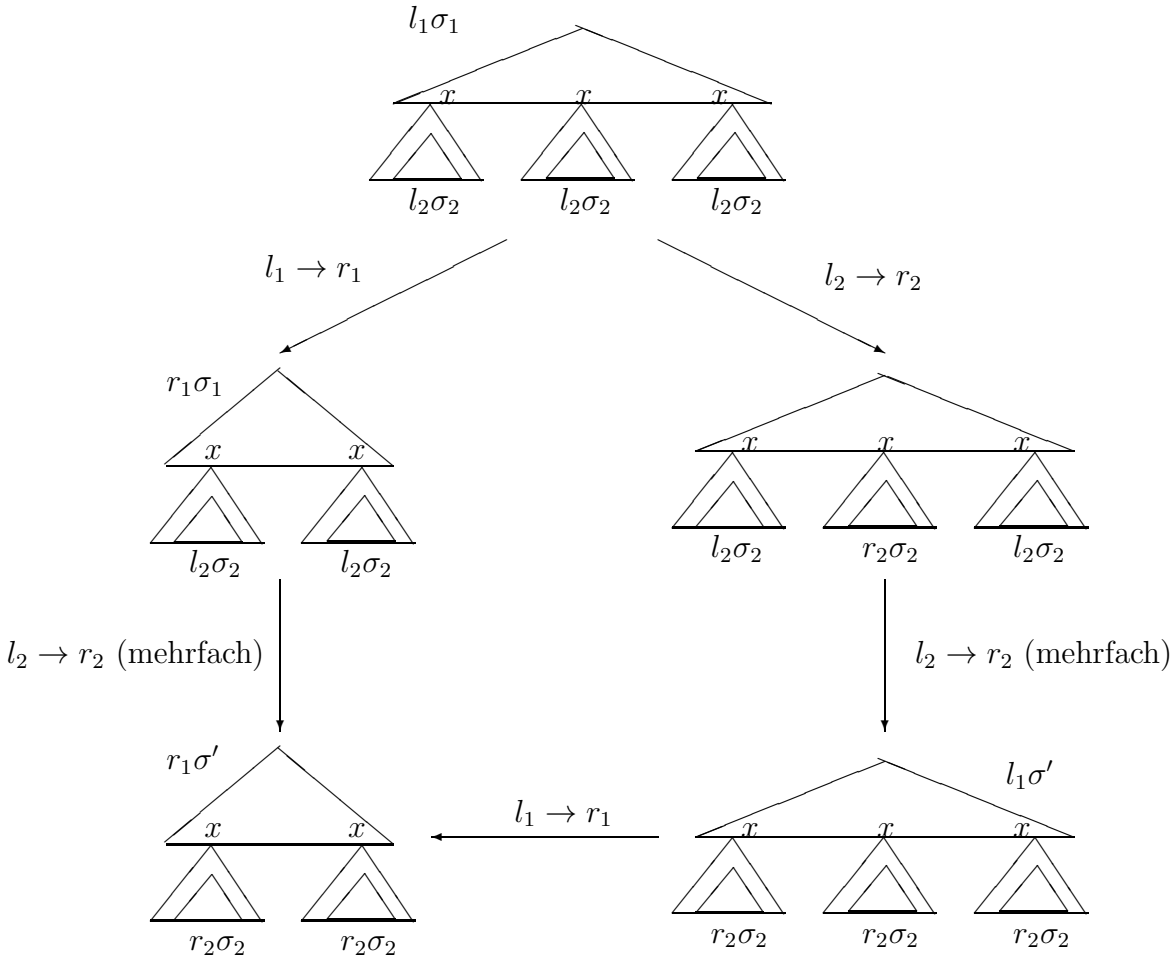
Hier wird also der Teilterm $p|_\pi$ zwar von l_2 gematcht, aber l_1 "versteckt" diesen Teil des Terms in einer Variable und "trägt" ihn bei der Ersetzung von $l_1\sigma_1$ durch $r_1\sigma_1$ "weiter". Die Regel $l_2 \to r_2$ kann daher nach dieser Ersetzung immer noch angewendet werden.

Genauer folgt in diesem Fall also, dass l_1 an einer Stelle π' eine Variable x hat und π unterhalb von π' liegt ($\pi = \pi'$ ist ebenfalls möglich). Es gilt also $\pi = \pi'\pi''$ mit $\pi' \in Occ(l_1)$ und $l_1|_{\pi'} \in \mathcal{V}$ (hierbei ist $\pi'' = \epsilon$ möglich). Diese Situation lässt sich durch folgendes Bild veranschaulichen:



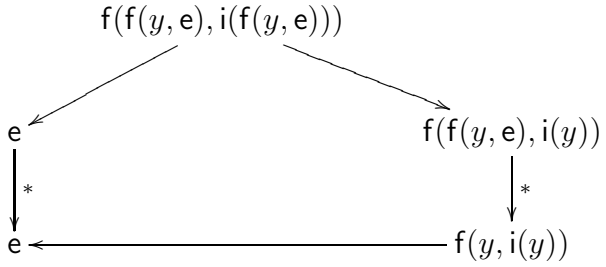
Der Redex $l_2\sigma_2$ ist demnach durch die Anwendung der Substitution σ_1 auf x entstanden, denn $x\sigma_1|_{\pi''} = l_2\sigma_2$. Der Term $l_1\sigma_1$ enthält also den Redex $l_2\sigma_2$ unterhalb aller Stellen, an denen in l_1 die Variable x auftrat. Die Regel $l_2 \to r_2$ ist daher so oft anwendbar, wie x in l_1 vorkam. Wenn x n-mal in l_1 auftrat, so entsteht bei (einmaliger) Anwendung von $l_2 \to r_2$ auf unseren ursprünglichen Term $p|_{\pi_1} = l_1\sigma_1$ ein Term $t|_{\pi_1}$, auf den die Regel $l_2 \to r_2$ noch $(n-1)$-mal angewendet werden muss. Hierdurch ergibt sich schließlich ein Term $l_1\sigma'$, auf den die andere Regel $l_1 \to r_1$ angewendet werden kann. Falls r_1 die Variable x m-mal enthielt, so muss die Regel $l_2 \to r_2$ nun auch m-mal auf den Term $s|_{\pi_1} = r_1\sigma_1$ angewendet werden, der durch Anwendung von $l_1 \to r_1$ aus $p|_{\pi_1} = l_1\sigma_1$ entstand. Auf diese Weise lassen sich $s|_{\pi_1}$ und $t|_{\pi_1}$ zu $r_1\sigma'$ zusammenführen.

Auch in diesem Fall sind die entstehenden Terme also stets zusammenführbar, aber hierzu können mehrere Reduktionsschritte notwendig sein, da l_1 und r_1 mehrfache Vorkommen der Variablen x enthalten können. Insgesamt erhält man folgende Situation (hier ist ein Beispiel dargestellt, in dem x dreimal in l_1 und zweimal in r_1 auftritt):



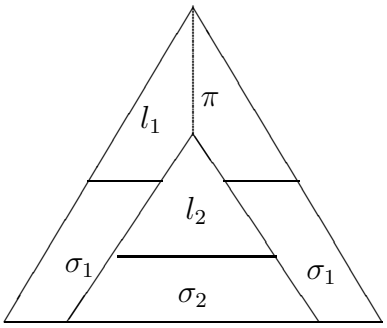
Solche Situationen bezeichnet man als "unkritische Überlappung", da hier die beiden Terme wiederum immer zusammengeführt werden können. Für die Untersuchung der lokalen Konfluenz brauchen wir solche Situationen daher nicht zu betrachten.

Zur Illustration dient wieder unser Gruppen-TES und die kritische Situation $\mathsf{e} \leftarrow_{\mathcal{R}} \mathsf{f}(\mathsf{f}(y,\mathsf{e}),\mathsf{i}(\mathsf{f}(y,\mathsf{e}))) \to_{\mathcal{R}} \mathsf{f}(\mathsf{f}(y,\mathsf{e}),\mathsf{i}(y))$. Hier ist $\pi_1 = \epsilon$ und $\pi_2 = 21$. Der Redex $l_2\sigma_2 = \mathsf{f}(y,\mathsf{e})$ entstand, indem die Variable x der ersten linken Seite $l_1 = \mathsf{f}(x,\mathsf{i}(x))$ durch $\mathsf{f}(y,\mathsf{e})$ instantiiert wurde. Die beiden Terme sind zusammenführbar, indem man alle verbliebenen instantiierten Vorkommen von x mit der Regel $l_2 \to r_2$ (d.h., $\mathsf{f}(x,\mathsf{e}) \to x$) reduziert. So ergibt sich



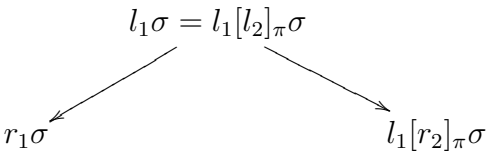
Fall 2.2: $l_2\sigma_2$ **liegt nicht im Substitutionsteil, d.h.,** $\pi \in Occ(l_1)$ **mit** $l_1|_\pi \notin \mathcal{V}$

Zur Veranschaulichung dient folgendes Bild:



Dieser Fall wird als *kritische Überlappung* bezeichnet. Wie aus der Betrachtung der anderen Fälle deutlich wurde, sind dies die einzigen Situationen, die man für die Untersuchung der lokalen Konfluenz betrachten muss, denn nur hier kann der Fall auftreten, dass die resultierenden Terme tatsächlich nicht zusammenführbar sind.

Allgemein entsteht ein solcher Fall immer dann, wenn es zwei Regeln $l_1 \to r_1$ und $l_2 \to r_2$ (die nicht notwendigerweise verschieden sein müssen), Substitutionen σ_1 und σ_2 und eine Stelle $\pi \in Occ(l_1)$ gibt, so dass $l_1|_\pi \notin \mathcal{V}$ und $l_1|_\pi\sigma_1 = l_2\sigma_2$ gilt. Wenn wir die Variablen in $l_1 \to r_1$ und in $l_2 \to r_2$ so umbenennen, dass sie paarweise disjunkt sind, so können wir $\sigma_1 = \sigma_2$ wählen. Eine kritische Überlappung entsteht also dann, wenn es zwei linke Regelseiten l_1, l_2 und eine Substitution σ gibt, die l_2 und einen Nicht-Variablen-Teilterm $l_1|_\pi$ identisch macht (d.h., $l_1|_\pi\sigma = l_2\sigma$). Mit anderen Worten, σ ist ein *Unifikator* von $l_1|_\pi$ und $l_2\sigma$. Auf diese Weise entsteht die kritische Situation



Es genügt, hierbei jeweils nur *allgemeinste* Unifikatoren von linken Regelseiten l_2 und Nicht-Variablen-Teiltermen $l_1|_\pi$ zu betrachten, denn wenn die jeweils durch einen Auswertungsschritt entstehenden Terme $r_1\sigma$ und $l_1[r_2]_\pi\sigma$ bei der allgemeineren Substitution σ zusammenführbar sind, dann gilt dies auch bei jeder Substitution, die spezieller als σ ist (aufgrund der Abgeschlossenheit der Ersetzungsrelation unter Substitutionen). Wir müssen also nur alle diejenigen Paare $\langle r_1\sigma, l_1[r_2]_\pi\sigma\rangle$ auf Zusammenführbarkeit überprüfen, die mit solch einem Unifikator σ entstehen. Man bezeichnet diese Paare daher als *kritische Paare*.

Definition 5.2.4 (Kritisches Paar [KB70]) *Sei $\mathcal{R}$ ein TES mit $l_1 \to r_1, l_2 \to r_2 \in \mathcal{R}$. Hierbei seien die Variablen der beiden Regeln so umbenannt, dass $\mathcal{V}(l_1) \cap \mathcal{V}(l_2) = \varnothing$ gilt. Sei $\pi \in Occ(l_1)$ mit $l_1|_\pi \notin \mathcal{V}$ und sei σ allgemeinster Unifikator von $l_1|_\pi$ und l_2. Dann ist $\langle r_1\sigma, l_1[r_2]_\pi\sigma\rangle$ ein* kritisches Paar *von $\mathcal{R}$ (engl. critical pair). Die beiden Regeln $l_1 \to r_1, l_2 \to r_2$ dürfen (bis auf Variablenumbenennung) identisch sein. In diesem Fall betrachtet man jedoch nur Überlappungen mit $\pi \neq \epsilon$. Die Menge der kritischen Paare eines TES $\mathcal{R}$ wird mit $CP(\mathcal{R})$ bezeichnet.*

Jedes (endliche) TES hat nur endlich viele kritische Paare, da man immer nur die (endlich vielen) linken Regelseiten auf Überlappungen überprüfen muss. $CP(\mathcal{R})$ kann durch Verwendung des Unifikationsalgorithmus UNIFY aus dem vorigen Abschnitt automatisch berechnet werden.

Beispiel 5.2.5 Wir betrachten wieder das Gruppenbeispiel.

$$\begin{aligned} \mathsf{f}(x, \mathsf{f}(y, z)) &\to \mathsf{f}(\mathsf{f}(x, y), z) \\ \mathsf{f}(x, \mathsf{e}) &\to x \\ \mathsf{f}(x, \mathsf{i}(x)) &\to \mathsf{e} \end{aligned}$$

Die erste und die zweite Regel bilden zusammen ein kritisches Paar. Die linke Seite $\mathsf{f}(x', \mathsf{e})$ der (variablenumbenannten) zweiten Regel unifiziert mit dem Teilterm $\mathsf{f}(y, z)$ der ersten linken Regelseite $\mathsf{f}(x, \mathsf{f}(y, z))$. Der mgu ist $\{x'/y, z/\mathsf{e}\}$. Dies entspricht dem Term $\mathsf{f}(x, \mathsf{f}(y, \mathsf{e}))$, der mit der ersten Regel zu $\mathsf{f}(\mathsf{f}(x, y), \mathsf{e})$ und mit der zweiten Regel zu $\mathsf{f}(x, y)$ reduziert werden kann. So ergibt sich das kritische Paar

$$\langle \mathsf{f}(\mathsf{f}(x, y), \mathsf{e}), \mathsf{f}(x, y) \rangle. \tag{5.7}$$

Die erste und die dritte Regel ergeben zusammen ebenfalls ein kritisches Paar, das durch die Unifikation der linken Seite $\mathsf{f}(x', \mathsf{i}(x'))$ der (variablenumbenannten) dritten Regel mit dem Teilterm $\mathsf{f}(y, z)$ der ersten linken Regelseite $\mathsf{f}(x, \mathsf{f}(y, z))$ entsteht. Hier lautet der mgu $\{x'/y, z/\mathsf{i}(y)\}$. Dies entspricht dem Term $\mathsf{f}(x, \mathsf{f}(y, \mathsf{i}(y)))$, der mit der ersten Regel zu $\mathsf{f}(\mathsf{f}(x, y), \mathsf{i}(y))$ und mit der dritten Regel zu $\mathsf{f}(x, \mathsf{e})$ reduziert werden kann. Man erhält

$$\langle \mathsf{f}(\mathsf{f}(x, y), \mathsf{i}(y)), \mathsf{f}(x, \mathsf{e}) \rangle. \tag{5.8}$$

Schließlich erhält man durch Überlappung der ersten Regel mit sich selbst ein weiteres kritisches Paar. Bei den Regeln $\mathsf{f}(x, \mathsf{f}(y, z)) \to \mathsf{f}(\mathsf{f}(x, y), z)$ und $\mathsf{f}(x', \mathsf{f}(y', z')) \to \mathsf{f}(\mathsf{f}(x', y'), z')$ unifiziert die linke Regelseite $\mathsf{f}(x, \mathsf{f}(y, z))$ nämlich mit dem Teilterm $\mathsf{f}(y', z')$. Dadurch erhält man den Term $\mathsf{f}(x', \mathsf{f}(x, \mathsf{f}(y, z)))$ und das kritische Paar

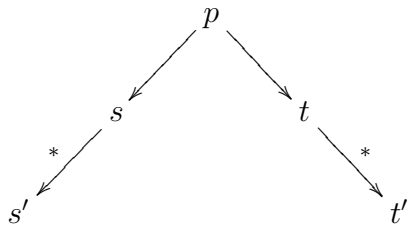
$$\langle \mathsf{f}(\mathsf{f}(x', x), \mathsf{f}(y, z)), \mathsf{f}(x', \mathsf{f}(\mathsf{f}(x, y), z)) \rangle \tag{5.9}$$

Aufgrund der vorangehenden Analyse ergibt sich, dass man für die lokale Konfluenz nur die kritischen Paare auf Zusammenführbarkeit untersuchen muss. Dieser Satz gilt unabhängig davon, ob $\mathcal{R}$ terminiert [Hue80].

Satz 5.2.6 (Kritisches-Paar-Lemma [KB70]) *Sei $\mathcal{R}$ ein TES. Dann ist $\mathcal{R}$ genau dann lokal konfluent, wenn alle seine kritischen Paare zusammenführbar sind.*

Beweis. Der Beweis folgt aus den vorangegangenen Überlegungen, da Situationen $s \leftarrow_{\mathcal{R}} p \to_{\mathcal{R}} t$, die keinem kritischen Paar entsprechen, immer zusammenführbar sind. □

Bei terminierenden TESen $\mathcal{R}$ kann die Zusammenführbarkeit der kritischen Paare leicht überprüft werden. Zu jedem kritischen Paar $\langle s, t \rangle$ berechnet man beliebige Normalformen s' und t' der Terme s und t, d.h., $s \to_{\mathcal{R}}^* s'$ und $t \to_{\mathcal{R}}^* t'$. Sofern die erhaltenen Normalformen s' und t' identisch sind, so ist das kritische Paar zusammenführbar. Falls die Normalformen nicht identisch sind, so ist das TES nicht konfluent, denn es existiert nach Konstruktion der kritischen Paare ein p mit



und s' und t' sind verschiedene Normalformen und damit nicht zusammenführbar. Da bei terminierenden TESen Konfluenz und lokale Konfluenz nach dem Diamond Lemma (Satz 5.2.3) gleichbedeutend sind, ist das TES dann also auch nicht lokal konfluent. So ergibt sich der folgende Algorithmus zur Konfluenzüberprüfung.

Algorithmus CONFLUENCE($\mathcal{R}$)
Eingabe: Ein terminierendes TES $\mathcal{R}$.
Ausgabe: "*True*", falls $\mathcal{R}$ konfluent ist und "*False*" sonst.

1. Berechne alle kritischen Paare $CP(\mathcal{R})$ von $\mathcal{R}$ (durch Verwendung von UNIFY).
2. Falls $CP(\mathcal{R}) = \varnothing$, dann gib "*True*" aus und breche ab.
3. Wähle $\langle s, t\rangle \in CP(\mathcal{R})$.
4. Reduziere s und t so lange wie möglich.
   Auf diese Weise entstehen die Normalformen s' und t'.
5. Falls $s' \neq t'$, dann gib "*False*" aus und breche ab.
6. Setze $CP(\mathcal{R}) = CP(\mathcal{R}) \setminus \{\langle s, t\rangle\}$.
7. Gehe zu Schritt 2.

Satz 5.2.7 (Korrektheit des Konfluenzuntersuchungsalgorithmus)

(a) Der Algorithmus CONFLUENCE *terminiert und ist korrekt.*

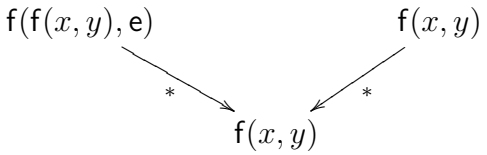
(b) Für terminierende TESe ist die Konfluenz entscheidbar.

Beweis.

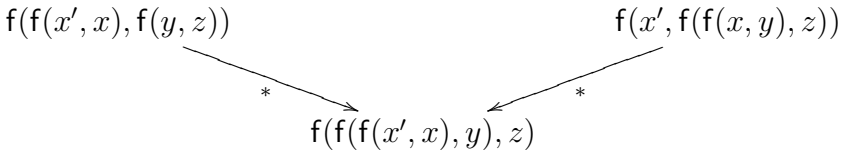
(a) Die Terminierung folgt daraus, dass UNIFY terminiert, dass $CP(\mathcal{R})$ immer endlich ist, dass die Berechnung der Normalformen terminiert (weil $\mathcal{R}$ terminiert) und dass $CP(\mathcal{R})$ in jedem Durchlauf der Schritte 2 - 7 kleiner wird. Falls der Algorithmus "*True*" ausgibt, so sind alle kritischen Paare zusammenführbar. Nach dem Kritischen-Paar-Lemma (Satz 5.2.6) folgt daraus die lokale Konfluenz und nach dem Diamond Lemma (Satz 5.2.3) die Konfluenz. Falls der Algorithmus "*False*" ausgibt, so existiert $s' \leftarrow^*_{\mathcal{R}} p \rightarrow^*_{\mathcal{R}} t'$ für zwei verschiedene Normalformen s' und t' und somit ist $\mathcal{R}$ nicht konfluent.

(b) Die Entscheidbarkeit folgt daraus, dass CONFLUENCE ein Entscheidungsalgorithmus ist (da auch die Bestimmung von allgemeinsten Unifikatoren berechenbar ist). □

Beispiel 5.2.8 Um die Konfluenz in dem Gruppenbeispiel zu untersuchen, muss man die kritischen Paare (5.7) - (5.9) auf Zusammenführbarkeit untersuchen. Hierzu reduzieren wir jeweils die Terme zu einer beliebigen Normalform. Bei dem ersten kritischen Paar (5.7) ergibt sich:



Dieses Paar ist also zusammenführbar. Das zweite kritische Paar (5.8) ist hingegen nicht zusammenführbar. Es lautet $\langle \mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)),\mathsf{f}(x,\mathsf{e})\rangle$. Der Term $\mathsf{f}(x,\mathsf{e})$ kann hierbei noch zu seiner Normalform x reduziert werden, aber da die Normalformen $\mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y))$ und x verschieden sind, erkennt man hier die Nicht-Konfluenz des Beispiels. Das dritte kritische Paar (5.9) ist hingegen wieder zusammenführbar:



Der Algorithmus CONFLUENCE würde also aufgrund des nicht-zusammenführbaren kritischen Paars (5.8) feststellen, dass das TES nicht konfluent ist.

Es ist uns also gelungen, die Frage der Eindeutigkeit von Berechnungen auf die Betrachtung von nur endlich vielen Termpaaren (den kritischen Paaren) zu reduzieren. Diese lassen sich mit der in Abschnitt 5.1 vorgestellten Technik der Unifikation berechnen und auf diese Weise lässt sich die Konfluenz eines (terminierenden) TES automatisch überprüfen. Die in diesem Kapitel vorgestellten Ergebnisse (Unifikation, Diamond Lemma, Kritisches-Paar-Lemma) haben auch über den Bereich der Termersetzungssysteme hinaus Bedeutung, wie z.B. bei der Untersuchung der Eindeutigkeit anderer Berechnungsformalismen. Das Konzept der kritischen Paare ist darüber hinaus entscheidend für das im nächsten Kapitel vorgestellte Vervollständigungsverfahren.

5.3 Konfluenz ohne Terminierung

Das Verfahren aus dem vorigen Abschnitt dient dazu, die Konfluenz von *terminierenden* TESen zu entscheiden. Dies ist insbesondere für die funktionale Programmierung wichtig. Ein funktionales Programm ist im wesentlichen eine Sammlung von Termersetzungsregeln und es wäre wieder wünschenswert, wenn man unabhängig von der Auswertungsstrategie immer das gleiche Ergebnis erhalten würde. Natürlich ist es aber möglich, *nicht-terminierende* funktionale Programme zu schreiben. Daher benötigt man ein Kriterium, dass auch bei nicht-terminierenden Programmen die Konfluenz sicher stellt.

In diesem Abschnitt werden wir zeigen, dass sogenannte *orthogonale* TESe stets konfluent sind, auch wenn sie nicht terminieren. Da funktionale Programme orthogonalen TESen entsprechen, folgt daraus die Konfluenz funktionaler Programme.

Im vorigen Abschnitt wurde gezeigt, dass ein TES lokal konfluent ist gdw. all seine kritischen Paare zusammenführbar sind. Bei terminierenden TESen ist dies gleichbedeutend zur Konfluenz. Um ein von der Terminierung unabhängiges Konfluenzkriterium zu finden, liegt es daher nahe, TESe zu betrachten, die gar keine kritischen Paare haben.

Definition 5.3.1 (Nicht-überlappendes TES) *Ein TES ist* nicht-überlappend*, falls es keine kritischen Paare hat.*

Man könnte vermuten, dass nicht-überlappende TESe stets konfluent sind. Das folgende Gegenbeispiel zeigt aber, dass dies nicht der Fall ist.

Beispiel 5.3.2 Wir betrachten das folgende (nicht-terminierende) TES:

$$\begin{aligned} \mathsf{eq}(x, x) &\to \mathsf{true} \\ \mathsf{eq}(x, \mathsf{succ}(x)) &\to \mathsf{false} \\ \mathsf{a} &\to \mathsf{succ}(\mathsf{a}) \end{aligned}$$

Das TES hat keine kritischen Paare, d.h., es ist nicht-überlappend. Trotzdem ist es nicht konfluent, denn wir haben

$$\mathsf{eq}(\mathsf{a}, \mathsf{a}) \to \mathsf{true} \qquad \text{und}$$
$$\mathsf{eq}(\mathsf{a}, \mathsf{a}) \to \mathsf{eq}(\mathsf{a}, \mathsf{succ}(\mathsf{a})) \to \mathsf{false}.$$

Das Problem des obigen Gegenbeispiels ist, dass die ersten beiden Regeln nicht *linkslinear* sind, d.h., ihre linken Seiten enthalten jeweils zwei Vorkommen der Variablen x. Wenn man dies ausschließt, kommt man zum Begriff der *Orthogonalität*.

Definition 5.3.3 (Linkslinearität, Orthogonalität) *Ein Term ist* linear*, wenn er keine mehrfachen Vorkommen derselben Variablen enthält. Ein TES ist* linkslinear*, falls die linken Seiten aller Regeln linear sind. Ein TES ist* orthogonal*, falls es nicht-überlappend und linkslinear ist.*

Wir wollen nun zeigen, dass alle orthogonalen TESe konfluent sind, unabhängig von ihrer Terminierung. Mit dem Kritischen-Paar-Lemma (Satz 5.2.6) sind nicht-überlappende TESe natürlich lokal konfluent. Hieraus folgt aber bei nicht-terminierenden TESen noch nicht die Konfluenz.

Wir führen daher nun eine noch stärkere Form von lokaler Konfluenz ein, die sogenannte *starke Konfluenz*.[1] Aus dieser folgt dann tatsächlich stets die Konfluenz, selbst bei nicht-terminierenden TESen, vgl. Satz 5.3.6.

Definition 5.3.4 (Starke Konfluenz) *Eine Relation $\to$ über einer Menge M heißt* stark konfluent*, wenn für alle $s, t, p \in M$ gilt:*

Wenn $p \to s$ und $p \to t$,
dann existiert ein $q \in M$ mit $s \to^= q$ und $t \to^= q$.

Hierbei bezeichnet $\to^=$ die reflexive Hülle von $\to$, d.h. es gilt $s \to^= q$ gdw. $s \to q$ oder $s = q$. Ein TES $\mathcal{R}$ heißt stark konfluent*, wenn die Ersetzungsrelation $\to_\mathcal{R}$ stark konfluent ist.*

Die starke Konfluenz unterscheidet sich also von der lokalen Konfluenz dadurch, dass Indeterminismen nach einem Schritt auch wieder in jeweils höchstens einem Schritt zusammenführbar sind.

[1] In der Literatur findet sich meist eine etwas andere Definition der "starken Konfluenz". Dort muss aus $p \to s$ und $p \to t$ folgen, dass ein q mit $s \to^* q$ und $t \to^= q$ existiert.

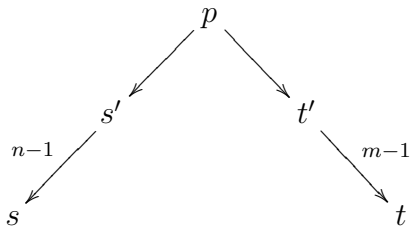
Beispiel 5.3.5 Das TES $\{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{c}, \mathsf{a} \to \mathsf{b}, \mathsf{c} \to \mathsf{b}\}$ ist stark konfluent und damit auch konfluent (aber nicht terminierend). Das TES $\{\mathsf{f}(x) \to \mathsf{g}(x, x), \mathsf{a} \to \mathsf{b}\}$ ist orthogonal, terminierend und konfluent, aber nicht stark konfluent. Es gilt nämlich $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b})$ und $\mathsf{f}(\mathsf{a}) \to \mathsf{g}(\mathsf{a}, \mathsf{a})$, aber um $\mathsf{f}(\mathsf{b})$ und $\mathsf{g}(\mathsf{a}, \mathsf{a})$ zusammenzuführen, muss man wie folgt vorgehen: $\mathsf{f}(\mathsf{b}) \to \mathsf{g}(\mathsf{b}, \mathsf{b})$ und $\mathsf{g}(\mathsf{a}, \mathsf{a}) \to^2 \mathsf{g}(\mathsf{b}, \mathsf{b})$. Dies bedeutet also, dass auch orthogonale TESe nicht zwangsläufig stark konfluent sind.

Der folgende Satz beweist nun, dass aus starker Konfluenz tatsächlich die Konfluenz folgt.

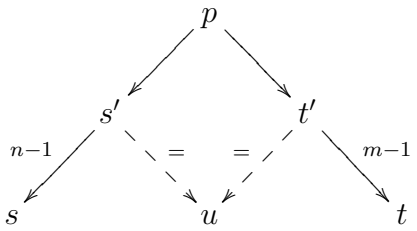
Satz 5.3.6 (Starke Konfluenz impliziert Konfluenz) *Sei $\to$ eine stark konfluente Relation über einer Menge M. Dann ist $\to$ konfluent.*

Beweis. Sei $p \to^n s$ und $p \to^m t$. Wir zeigen durch Induktion über $n + m$, dass es dann ein q gibt, so dass $s \to^{\leq m} q$ und $t \to^{\leq n} q$ gilt. Hierbei bezeichnet "$\to^{\leq m}$" eine höchstens m-fache Anwendung von $\to$.

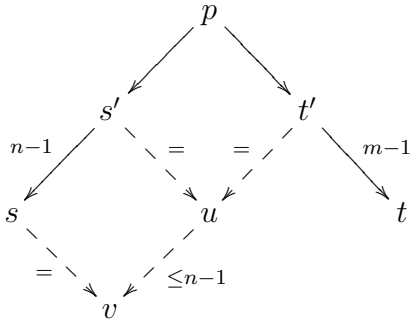
Falls $n = 0$ ist, so wählen wir $q = t$ und erhalten $s = p \to^m t = q$ und $t \to^0 t = q$. Der Fall $m = 0$ ist analog. Falls schließlich $n > 0$ und $m > 0$ ist, so existieren also s' und t' mit



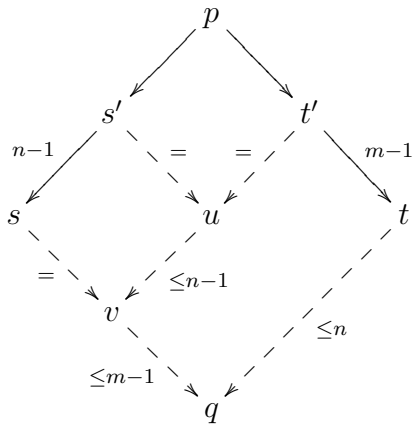
Aufgrund der starken Konfluenz gibt es also ein Objekt u mit $s' \to^= u \leftarrow^= t'$, d.h.



Nun können wir die Induktionshypothese anwenden. Der Indeterminismus, der in s' startet, benötigt insgesamt höchstens $n - 1 + 1 = n$ Schritte und somit weniger als $n + m$. Deshalb existiert also ein v mit $s \to^= v \leftarrow^{\leq n-1} u$, d.h.



Nun wenden wir die Induktionshypothese noch einmal an. Der Indeterminismus, der in t' startet, benötigt insgesamt höchstens $1 + n - 1 + m - 1 = n + m - 1$ Schritte und somit weniger als $n + m$. Deshalb existiert also ein q mit $v \to^{\leq m-1} q \leftarrow^{\leq n} t$, d.h.



□

Bsp. 5.3.5 zeigt bereits, dass auch bei orthogonalen TESen $\mathcal{R}$ die Ersetzungsrelation $\to_{\mathcal{R}}$ nicht immer stark konfluent ist. Das Problem tritt dann auf, wenn eine linke Regelseite l_1 eine Variable x enthält und diese durch σ_1 so instantiiert wird, dass sie eine Instanz einer (ggf. anderen) linken Regelseite l_2 enthält vgl. Fall 2.1 im Beweis des Kritischen–Paar–Lemmas. Aufgrund der Linkslinearität enthält l_1 nur ein Vorkommen der Variablen x. Aber der Term r_1 muss nicht linear sein und kann daher m Vorkommen von x enthalten, wobei m eine beliebige natürliche Zahl ist. Um das entstehende kritische Paar zusammenzuführen, muss man daher $r_1\sigma_1$ m-mal reduzieren. Dies ist bei starker Konfluenz nicht erlaubt.

Wir definieren daher nun eine Variante der Ersetzungsrelation $\rightrightarrows_{\mathcal{R}}$, bei der Redexe an voneinander unabhängigen Stellen gleichzeitig (d.h. parallel) reduziert werden können. Wir werden dann zeigen, dass diese Relation bei orthogonalen TESen stark konfluent ist.

Definition 5.3.7 (Parallele Ersetzungsrelation) *Eine Menge* Π *von Stellen ist* parallel *gdw.* $\pi_1 \perp \pi_2$ *für alle* $\pi_1, \pi_2 \in \Pi$ *mit* $\pi_1 \neq \pi_2$ *gilt. Für ein TES* $\mathcal{R}$ *ist die* parallele Ersetzungsrelation $\rightrightarrows_{\mathcal{R}}$ *wie folgt definiert. Es gilt* $s \rightrightarrows_{\mathcal{R}} t$ *gdw. es eine Menge* $\Pi = \{\pi_1, \ldots, \pi_n\}$ *paralleler Stellen von* s *gibt (mit* $n \geq 0$*) und es für jedes* π_i *eine Regel* $l_i \to r_i \in \mathcal{R}$ *und eine Substitution* σ_i *gibt, so dass* $s|_{\pi_i} = l_i\sigma_i$ *und* $t = s[r_1\sigma_1]_{\pi_1} \ldots [r_n\sigma_n]_{\pi_n}$*. Wir schreiben dann auch* $s \rightrightarrows^{\Pi}_{\mathcal{R}} t$*.*

Beispiel 5.3.8 Wir betrachten wieder das TES $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{g}(x, x), \mathsf{a} \to \mathsf{b}\}$ aus Bsp. 5.3.5. Der Term $\mathsf{g}(\mathsf{a}, \mathsf{a})$ hat z.B. die parallelen Stellen $\Pi = \{1, 2\}$ und es gilt $\mathsf{g}(\mathsf{a}, \mathsf{a}) \rightrightarrows^{\Pi}_{\mathcal{R}} \mathsf{g}(\mathsf{b}, \mathsf{b})$. Der Indeterminismus aus Bsp. 5.3.5 lässt sich also in einem parallelen Ersetzungsschritt zusammenführen, d.h., $\rightrightarrows_{\mathcal{R}}$ ist stark konfluent.

Bevor wir beweisen, dass $\rightrightarrows_{\mathcal{R}}$ bei orthogonalen TESen $\mathcal{R}$ immer stark konfluent und damit insbesondere konfluent ist, zeigt das folgende Lemma den Zusammenhang zwischen der normalen Ersetzungsrelation $\to_{\mathcal{R}}$ und der parallelen Ersetzungsrelation $\rightrightarrows_{\mathcal{R}}$. Insbeson-

dere besagt das Lemma, dass aus der Konfluenz von $\rightrightarrows_{\mathcal{R}}$ dann auch die Konfluenz von $\to_{\mathcal{R}}$ folgt.

Lemma 5.3.9 ($\to_{\mathcal{R}}$ und $\rightrightarrows_{\mathcal{R}}$) *Sei $\mathcal{R}$ ein TES.*

(a) Es gilt $\to_{\mathcal{R}} \subseteq \rightrightarrows_{\mathcal{R}} \subseteq \to^*_{\mathcal{R}}$.

(b) $\to_{\mathcal{R}}$ ist konfluent gdw. $\rightrightarrows_{\mathcal{R}}$ konfluent ist.

Beweis.

(a) Wir schreiben $s \to^{\pi}_{\mathcal{R}} t$, falls π die Stelle des Redexes von s ist. Dann folgt aus $s \to^{\pi}_{\mathcal{R}} t$ jeweils $s \rightrightarrows^{\{\pi\}}_{\mathcal{R}} t$. Weiterhin folgt bei $\Pi = \{\pi_1, \ldots, \pi_n\}$ aus $s \rightrightarrows^{\Pi}_{\mathcal{R}} t$ offensichtlich $s \to^{\pi_1}_{\mathcal{R}} s_1 \to^{\pi_2}_{\mathcal{R}} s_2 \to^{\pi_3}_{\mathcal{R}} \ldots \to^{\pi_n}_{\mathcal{R}} s_n = t$, d.h. $s \to^*_{\mathcal{R}} t$.

(b) Wegen (a) gilt $\to^*_{\mathcal{R}} \subseteq \rightrightarrows^*_{\mathcal{R}} \subseteq (\to^*_{\mathcal{R}})^* = \to^*_{\mathcal{R}}$, d.h. $\to^*_{\mathcal{R}} = \rightrightarrows^*_{\mathcal{R}}$. Damit ergibt sich:

$$\begin{array}{ll} & \to_{\mathcal{R}} \text{ ist konfluent} \\ \text{gdw.} & \to^*_{\mathcal{R}} \text{ ist konfluent} \\ \text{gdw.} & \rightrightarrows^*_{\mathcal{R}} \text{ ist konfluent} \\ \text{gdw.} & \rightrightarrows_{\mathcal{R}} \text{ ist konfluent} \end{array}$$

□

Nun können wir das gewünschte Konfluenzkriterium beweisen.

Satz 5.3.10 (Orthogonalität impliziert Konfluenz) *Jedes orthogonale TES ist konfluent.*

Beweis. Sei $\mathcal{R}$ ein orthogonales TES. Wir zeigen, dass $\rightrightarrows_{\mathcal{R}}$ stark konfluent ist. Dann gilt:

$$\begin{array}{lll} & \rightrightarrows_{\mathcal{R}} \text{ ist stark konfluent} & \\ \curvearrowright & \rightrightarrows_{\mathcal{R}} \text{ ist konfluent} & \text{nach Satz 5.3.6} \\ \text{gdw.} & \to_{\mathcal{R}} \text{ ist konfluent} & \text{nach Lemma 5.3.9 (b)} \end{array}$$

Seien Π_1 und Π_2 parallele Mengen von Stellen des Terms p und sei $p \rightrightarrows^{\Pi_1} s$ und $p \rightrightarrows^{\Pi_2} t$. Wir teilen die Menge Π_1 wie folgt auf:

- A_1 sind die Stellen aus Π_1, die nicht unterhalb einer Stelle von Π_2 liegen, d.h., $A_1 = \{\pi \in \Pi_1 \mid \text{es existiert kein } \pi' \in \Pi_2 \text{ mit } \pi \geq_{\mathbb{N}^*} \pi'\}$.
- B_1 sind die Stellen aus Π_1, die echt unterhalb einer Stelle von Π_2 liegen, d.h., $B_1 = \{\pi \in \Pi_1 \mid \text{es existiert ein } \pi' \in \Pi_2 \text{ mit } \pi >_{\mathbb{N}^*} \pi'\}$.
- C sind die Stellen, die sowohl in Π_1 als auch in Π_2 liegen, d.h. $C = \Pi_1 \cap \Pi_2$.

Damit gilt also $\Pi_1 = A_1 \uplus B_1 \uplus C$. Analog dazu definieren wir A_2 und B_2 und erhalten $\Pi_2 = A_2 \uplus B_2 \uplus C$.

Die Menge $\Pi = A_1 \uplus A_2 \uplus C$ ist offensichtlich eine parallele Menge von Stellen von s und t und an allen Stellen, die oberhalb oder unabhängig von den Stellen in Π sind, haben s und t die gleichen Symbole. Es genügt also zu zeigen, dass es für alle $\pi \in \Pi$ einen Term $q|_\pi$ gibt mit $s|_\pi \rightrightarrows_\mathcal{R} q|_\pi$ und $t|_\pi \rightrightarrows_\mathcal{R} q|_\pi$. Wenn $\Pi = \{\pi_1, \ldots, \pi_n\}$ und $q = s[q|_{\pi_1}]_{\pi_1} \ldots [q|_{\pi_n}]_{\pi_n}$, dann gilt nämlich $s \rightrightarrows_\mathcal{R} q$ und $t \rightrightarrows_\mathcal{R} q$.

Wir betrachten zunächst den Fall $\pi \in C$. Damit folgt $p \to^\pi_\mathcal{R} s$ und $p \to^\pi_\mathcal{R} t$. Es existieren also zwei Regeln $l \to r$ und $l' \to r'$ (die bis auf Variablenumbenennung gleich sein können) sowie eine Substitution σ mit

- $p|_\pi = l\sigma = l'\sigma$
- $s|_\pi = r\sigma$
- $t|_\pi = r'\sigma$

Da $\mathcal{R}$ aber keine kritischen Paare besitzt, folgt $l = l'$ und $r = r'$. Damit erhält man also $s|_\pi = t|_\pi$. Falls man $q|_\pi = s|_\pi$ definiert, folgt damit auch $s|_\pi \rightrightarrows^\varnothing_\mathcal{R} q|_\pi$ und $t|_\pi \rightrightarrows^\varnothing_\mathcal{R} q|_\pi$.

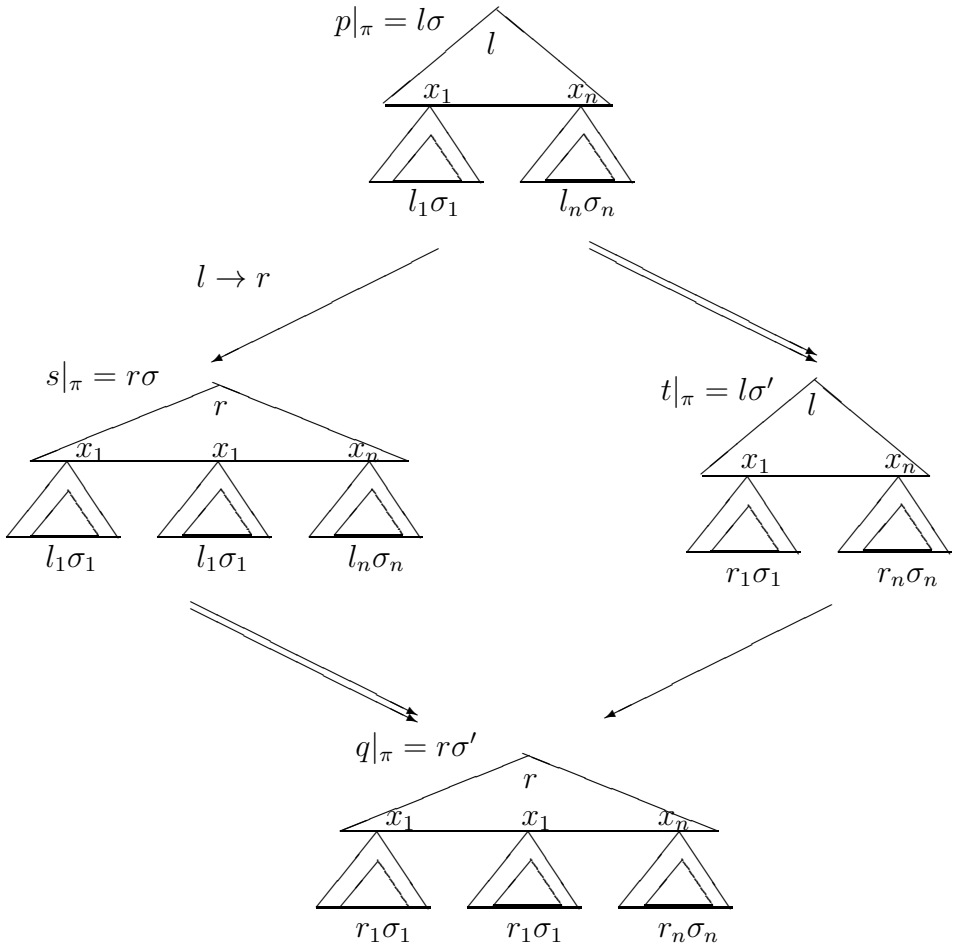
Nun betrachten wir den Fall $\pi \in A_1$. Der Fall $\pi \in A_2$ ist analog. Seien $\pi_1, \ldots, \pi_n$ die Positionen aus B_2, die echt unterhalb von π liegen, d.h., $\pi_i >_{\mathbb{N}^*} \pi$ für alle i. Anders ausgedrückt existieren also Stellen π'_i mit $\pi_i = \pi\,\pi'_i$ für alle i. Es existieren dann Regeln $l \to r$, $l_1 \to r_1$, $\ldots$, $l_n \to r_n$ und Substitutionen $\sigma, \sigma_1, \ldots, \sigma_n$ mit denen die jeweiligen Reduktionen durchgeführt werden. Es gilt also $p|_\pi \to^\varepsilon s|_\pi$ und $p|_\pi \rightrightarrows^{\{\pi'_1,\ldots,\pi'_n\}} t|_\pi$.

Da $\mathcal{R}$ keine kritischen Paare besitzt, liegen die Redexe $l_i\sigma_i$ im Substitutionsteil von $l\sigma$. Der Term l enthält also Variablen $x_1, \ldots, x_n$, so dass $l_i\sigma_i$ jeweils ein Teilterm von $x_i\sigma$ ist. Hierbei kann auch $x_i = x_j$ für $i \neq j$ gelten, d.h., ein $x_i\sigma$ kann mehrere Redexe $l_i\sigma_i$ und $l_j\sigma_j$ enthalten. Es existiert jeweils eine Stelle τ_i mit $(x_i\sigma)|_{\tau_i} = l_i\sigma_i$. Da l linear ist, tritt jedes x_i nur genau einmal in l auf. Die Reduktion $p|_\pi = l\sigma \rightrightarrows^{\{\pi'_1,\ldots,\pi'_n\}} t|_\pi$ bedeutet also, dass nur die Instantiierungen der x_i in l geändert werden. Wir definieren eine neue Substitution σ' als $x_i\sigma' = (x_i\sigma)[r_i\sigma_i]_{\tau_i}$ und $x\sigma' = x\sigma$ für alle Variablen $x \notin \{x_1, \ldots, x_n\}$. Dann gilt

- $p|_\pi = l\sigma$
- $s|_\pi = r\sigma$
- $t|_\pi = l\sigma'$

Wir definieren $q|_\pi = r\sigma'$. Dann gilt offensichtlich $t|_\pi \to^\varepsilon_\mathcal{R} q|_\pi$. Der Term r ist nicht unbedingt linear und er kann daher die Variablen x_i mehrfach enthalten. Für jede Stelle δ von r an der eine Variable x_i steht, enthalte Δ die Stelle $\delta\,\tau_i$. Es gilt nämlich $(r\sigma)|_{\delta\,\tau_i} = l_i\sigma_i$. Man erhält daher $s|_\pi = r\sigma \rightrightarrows^\Delta r\sigma' = q|_\pi$. Die Konstruktion ist im folgenden Diagramm veranschaulicht:[2]

[2]Hierbei betrachten wir den Spezialfall, in dem $x_i\sigma_i$ nur jeweils ein $l_i\sigma_i$ enthält.


$p|_\pi = l\sigma$
l
x_1
x_n
$l_1\sigma_1$
$l_n\sigma_n$
$l \to r$
$s|_\pi = r\sigma$
r
x_1
x_1
x_n
$l_1\sigma_1$
$l_1\sigma_1$
$l_n\sigma_n$
$t|_\pi = l\sigma'$
l
x_1
x_n
$r_1\sigma_1$
$r_n\sigma_n$
$q|_\pi = r\sigma'$
r
x_1
x_1
x_n
$r_1\sigma_1$
$r_1\sigma_1$
$r_n\sigma_n$

□

Kapitel 6

Vervollständigung von Termersetzungssystemen

Um das Wortproblem für ein Gleichungssystem $\mathcal{E}$ zu lösen, versuchen wir, ein zu $\mathcal{E}$ äquivalentes konvergentes TES $\mathcal{R}$ zu konstruieren. Damit ist das Wortproblem (mit Hilfe des Algorithmus WORTPROBLEM) entscheidbar, da dann $s \equiv_{\mathcal{E}} t$ genau dann gilt, wenn die $\mathcal{R}$-Normalformen von s und t identisch sind. In Abbildung 3.1 wurde deutlich, wie man zu $\mathcal{E}$ ein äquivalentes konvergentes TES erhält: Man überführt zunächst die Gleichungen aus $\mathcal{E}$ in Regeln, indem das Gleichheitszeichen $\equiv$ jeweils durch $\rightarrow$ oder durch $\leftarrow$ ersetzt wird. Dies muss so geschehen, dass die Terminierung des entstehenden TES $\mathcal{R}$ nachgewiesen werden kann. Um die Terminierung von TESen automatisch zu untersuchen, haben wir in Kapitel 4 die Technik der lexikographischen und der rekursiven Pfadordnung bzw. der rekursiven Pfadordnung mit Status kennen gelernt, die diese beiden Ordnungen verbindet. Anschließend wird die Konfluenz des entstandenen TES $\mathcal{R}$ wie in Kapitel 5 überprüft. Falls $\mathcal{R}$ sowohl terminiert als auch konfluent ist, so ist hiermit ein äquivalentes konvergentes TES gefunden und damit hat man (automatisch) ein Entscheidungsverfahren für $\mathcal{E}$ konstruiert. Ansonsten muss man nun in einem *Vervollständigungsschritt* (engl. *completion*) $\mathcal{R}$ um neue Regeln ergänzen und durchläuft dann die Terminierungs- und Konfluenzüberprüfung erneut. Diese Schritte können beliebig oft wiederholt werden. Das Ziel dieses Kapitels ist, geeignete Vervollständigungsprozeduren vorzustellen, d.h., Verfahren, die automatisch einen Entscheidungsalgorithmus zu einem gegebenen Gleichungssystem konstruieren. In Abschnitt 6.1 stellen wir einen ersten Vervollständigungsalgorithmus vor. Dieser wird dann in Abschnitt 6.2 verbessert, wobei wir eine ganze Klasse von Vervollständigungsalgorithmen vorstellen und Kriterien für ihre Korrektheit angeben. Schließlich zeigen wir in Abschnitt 6.3, dass sich der Vervollständigungsalgorithmus auch verwenden lässt, um Induktionsbeweise zu führen und dadurch Programme (bzw. Termersetzungssysteme) zu verifizieren.

6.1 Der grundlegende Vervollständigungsalgorithmus

Wir betrachten noch einmal unsere beiden Haupt-Beispiele.

Beispiel 6.1.1 Die folgenden beiden Gleichungen stellen einen Additionsalgorithmus dar.

$$\mathcal{E} = \{\mathsf{plus}(\mathcal{O}, y) \equiv y, \mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathsf{plus}(x, y))\}$$

Um ein zu $\mathcal{E}$ äquivalentes konvergentes TES zu konstruieren, beginnen wir mit einem TES $\mathcal{R}$, das durch Orientieren der Gleichungen entsteht. Wir wählen z.B.

$$\mathcal{R} = \{\mathsf{plus}(\mathcal{O}, y) \to y, \mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathsf{plus}(x, y))\}.$$

Nun muss die Terminierung von $\mathcal{R}$ gezeigt werden, was mit der RPOS (sogar der lexikographischen Pfadordnung) bei der Präzedenz $\mathsf{plus} \sqsupset \mathsf{succ}$ leicht möglich ist. Schließlich wird die Konfluenz überprüft. Da das System keine kritischen Paare hat, ist es konfluent und somit ist ein konvergentes zu $\mathcal{E}$ äquivalentes TES gefunden. Hiermit lassen sich nun beliebige Aussagen über $\mathcal{E}$ nachweisen bzw. widerlegen. Um beispielsweise $\mathsf{plus}(\mathsf{succ}(\mathsf{succ}(\mathcal{O})), x) \equiv_{\mathcal{E}} \mathsf{plus}(\mathsf{succ}(\mathcal{O}), \mathsf{succ}(x))$ zu überprüfen, werden die beiden Terme mit $\mathcal{R}$ zu Normalformen reduziert. Da beide die Normalform $\mathsf{succ}(\mathsf{succ}(x))$ haben, gilt diese Gleichung.

Beispiel 6.1.2 Nun betrachten wir wieder die Gleichungen $\mathcal{E}$ zur Axiomatisierung von Gruppen.

$$\mathcal{E} = \{\mathsf{f}(x, \mathsf{f}(y, z)) \equiv \mathsf{f}(\mathsf{f}(x, y), z), \mathsf{f}(x, \mathsf{e}) \equiv x, \mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e}\}.$$

Bei Ersetzung des Gleichheitszeichens $\equiv$ durch einen Pfeil $\to$ erhält man ein zu $\mathcal{E}$ äquivalentes TES $\mathcal{R}_0$:

$$\begin{array}{rrcll} \mathcal{R}_0: & \mathsf{f}(x, \mathsf{f}(y, z)) & \to & \mathsf{f}(\mathsf{f}(x, y), z) & (G1) \\ & \mathsf{f}(x, \mathsf{e}) & \to & x & (G2) \\ & \mathsf{f}(x, \mathsf{i}(x)) & \to & \mathsf{e} & (G3) \end{array}$$

Die Terminierung des TES $\mathcal{R}_0$ lässt sich wieder mit der RPOS zeigen (bzw. mit der lexikographischen Pfadordnung, wobei f den Status $\langle 2, 1\rangle$ hat und man $\mathsf{f} \sqsupset \mathsf{e}$ in der Präzedenz setzt). Zur Überprüfung der Konfluenz werden die kritischen Paare von $\mathcal{R}_0$ berechnet, vgl. Bsp. 5.2.5:

$$\begin{array}{ll} \langle \mathsf{f}(\mathsf{f}(x, y), \mathsf{e}), \mathsf{f}(x, y)\rangle & (5.7) \\ \langle \mathsf{f}(\mathsf{f}(x, y), \mathsf{i}(y)), \mathsf{f}(x, \mathsf{e})\rangle & (5.8) \\ \langle \mathsf{f}(\mathsf{f}(x', x), \mathsf{f}(y, z)), \mathsf{f}(x', \mathsf{f}(\mathsf{f}(x, y), z))\rangle & (5.9) \end{array}$$

Nun wird überprüft, ob sich die kritischen Paare zusammenführen lassen, vgl. Bsp. 5.2.8. Genauer werden die Terme in den kritischen Paaren zu Normalformen mit $\mathcal{R}_0$ reduziert. So ergibt sich

$$\begin{array}{ll} \langle \mathsf{f}(x, y), \mathsf{f}(x, y)\rangle & \text{aus } (5.7) \\ \langle \mathsf{f}(\mathsf{f}(x, y), \mathsf{i}(y)), x\rangle & \text{aus } (5.8) \\ \langle \mathsf{f}(\mathsf{f}(\mathsf{f}(x', x), y), z), \mathsf{f}(\mathsf{f}(\mathsf{f}(x', x), y), z)\rangle & \text{aus } (5.9) \end{array}$$

Da (5.8) nicht zusammenführbar ist, ist das TES $\mathcal{R}_0$ nicht konfluent.

Die kritischen Paare aus Abschnitt 5.2 identifizieren die problematischen Termpaare, an denen das Scheitern der (lokalen) Konfluenz liegt. Die Grundidee der Vervollständigung (von Knuth und Bendix [KB70]) beruht daher darauf, aus den kritischen Paaren neue Regeln zu generieren. Man behandelt daher die (zu Normalformen reduzierten) kritischen Paare wie zusätzliche Gleichungen aus $\mathcal{E}$, die nun ebenfalls orientiert und in das TES eingefügt werden müssen. Hierbei können natürlich bereits zusammenführbare kritische Paare ignoriert werden. Im Gruppenbeispiel muss man also nun das bisherige TES $\mathcal{R}_0$ um eine Regel ergänzen, die dem (zu Normalformen reduzierten) kritischen Paar (5.8)

entspricht. Dabei soll natürlich die Terminierung erhalten bleiben, d.h., wir verwenden die RPOS aus dem Terminierungsbeweis von $\mathcal{R}_0$, um zu entscheiden, in welcher Richtung dieses Paar orientiert werden soll. Selbstverständlich geht man hierbei wieder so vor, dass man möglichst viele Festlegungen der RPOS (wie Präzedenz und Status) solange wie möglich offen lässt und erst nach und nach bei der Orientierung der Paare diese Festlegungen trifft. In unserem Beispiel muss das Paar von links nach rechts orientiert werden, denn dann gilt weiterhin $l \succ_{rpos} r$ für alle Regeln. Auf diese Weise entsteht das TES $\mathcal{R}_1$:

$$\begin{array}{lrcll} \mathcal{R}_1: & \mathsf{f}(x,\mathsf{f}(y,z)) & \to & \mathsf{f}(\mathsf{f}(x,y),z) & (G1)\\ & \mathsf{f}(x,\mathsf{e}) & \to & x & (G2)\\ & \mathsf{f}(x,\mathsf{i}(x)) & \to & \mathsf{e} & (G3)\\ & \mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) & \to & x & (G4) \end{array}$$

Man erkennt, dass durch (reduzierte) kritische Paare hilfreiche "Lemmata" automatisch generiert werden, die zum einen interessante Aussagen über die Gleichungstheorie darstellen und die zum anderen für spätere Beweise hilfreich sind.

Die Terminierung von $\mathcal{R}_1$ ist nach Konstruktion sicher gestellt. Ebenso gilt $\leftrightarrow^*_{\mathcal{R}_1} = \leftrightarrow^*_{\mathcal{R}_0}$ und somit ist auch $\mathcal{R}_1$ äquivalent zu $\mathcal{E}$. Der Grund ist, dass für alle kritischen Paare $\langle s,t\rangle$ aus $\mathcal{R}_0$ jeweils $s \leftarrow_{\mathcal{R}_0} p \to_{\mathcal{R}_0} t$ gilt. Für die Normalformen s' und t' von s und t gilt daher $s' \leftarrow^*_{\mathcal{R}_0} p \to^*_{\mathcal{R}_0} t'$. Die neuen Regeln in $\mathcal{R}_1$ haben entweder die Form $s' \to t'$ oder $t' \to s'$. In beiden Fällen sind die Terme auf den beiden Seiten der Regeln aber $\leftrightarrow^*_{\mathcal{R}_0}$ äquivalent. Somit folgt $\leftrightarrow^*_{\mathcal{R}_1} = \leftrightarrow^*_{\mathcal{R}_0}$.

$\mathcal{R}_1$ "beschreibt" also die gleiche Theorie wie $\mathcal{R}_0$, es terminiert weiterhin und die Gründe für die Nicht-Konfluenz von $\mathcal{R}_0$ wurden in $\mathcal{R}_1$ beseitigt. Durch die neu hinzugefügten Regeln sind nämlich alle kritischen Paare zwischen den $\mathcal{R}_0$-Regeln zusammenführbar. Allerdings können durch die neuen Regeln neue kritische Paare entstehen. Diese muss man berechnen und auf Zusammenführbarkeit überprüfen, um die Konfluenz von $\mathcal{R}_1$ zu untersuchen. In unserem Beispiel ergeben sich neben (5.7) - (5.9) folgende kritische Paare von $\mathcal{R}_1$:

$$\begin{array}{rl} \langle \mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),\mathsf{i}(y)),\mathsf{f}(x',x)\rangle & \text{aus } (G1) \text{ und } (G4), \text{ zusammenführbar}\\ \langle x,\mathsf{f}(\mathsf{f}(\mathsf{f}(x,y),z),\mathsf{i}(\mathsf{f}(y,z)))\rangle & \text{aus } (G4) \text{ und } (G1), \text{ nicht zusammenführbar}\\ \langle x,\mathsf{f}(x,\mathsf{i}(\mathsf{e}))\rangle & \text{aus } (G4) \text{ und } (G2), \text{ nicht zusammenführbar}\\ \langle x,\mathsf{f}(\mathsf{e},\mathsf{i}(\mathsf{i}(x)))\rangle & \text{aus } (G4) \text{ und } (G3), \text{ nicht zusammenführbar}\\ \langle \mathsf{f}(x,y),\mathsf{f}(x,\mathsf{i}(\mathsf{i}(y)))\rangle & \text{aus } (G4) \text{ und } (G4), \text{ nicht zusammenführbar} \end{array}$$

Nun wird das Verfahren wiederholt. Hierzu normalisiert man die obigen kritischen Paare (mit $\mathcal{R}_1$) und fügt die entsprechend orientierten Paare als neue Regeln zu $\mathcal{R}_1$ hinzu (bis auf die Paare, bei denen die Normalisierung beider Terme zum gleichen Ergebnis führt). Auf diese Weise entsteht $\mathcal{R}_2$.

$$\begin{array}{lrcll} \mathcal{R}_2: & \mathsf{f}(x,\mathsf{f}(y,z)) & \to & \mathsf{f}(\mathsf{f}(x,y),z) & (G1)\\ & \mathsf{f}(x,\mathsf{e}) & \to & x & (G2)\\ & \mathsf{f}(x,\mathsf{i}(x)) & \to & \mathsf{e} & (G3)\\ & \mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) & \to & x & (G4)\\ & \mathsf{f}(\mathsf{f}(\mathsf{f}(x,y),z),\mathsf{i}(\mathsf{f}(y,z))) & \to & x & \\ & \mathsf{f}(x,\mathsf{i}(\mathsf{e})) & \to & x & \\ & \mathsf{f}(\mathsf{e},\mathsf{i}(\mathsf{i}(x))) & \to & x & \\ & \mathsf{f}(x,\mathsf{i}(\mathsf{i}(y))) & \to & \mathsf{f}(x,y) & \end{array}$$

Das Verfahren wird nun so lange wiederholt, bis keine neuen Regeln mehr entstehen, d.h., bis alle kritischen Paare zusammenführbar sind. Wie in Abschnitt 3.3 erwähnt, erhält man so eine Folge $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$ von TESen, die alle zu $\mathcal{E}$ äquivalent sind. Wie dort erläutert wurde, gibt es 3 Möglichkeiten für diesen Prozess:

1. Man könnte nach einigen Iterationen ein TES $\mathcal{R}_n$ erhalten, dessen kritische Paare alle zusammenführbar sind. Dann endet das Verfahren mit Erfolg, denn $\mathcal{R}_n$ ist ein konvergentes und zu $\mathcal{E}$ äquivalentes TES.

2. Es könnte sein, dass man bei einem TES $\mathcal{R}_n$ in der Folge die Terminierung nicht nachweisen kann. (Dies kann daran liegen, dass es wirklich nicht terminiert oder dass unser hinreichendes Kriterium der Terminierungsüberprüfung mit der RPOS nur die Terminierung nicht nachweisen kann.) Nun bricht man mit Misserfolg ("*Fail*") ab. Immerhin erkennt man in diesem Fall den Misserfolg und könnte z.B. mit einem verbesserten Terminierungstest einen neuen Vervollständigungsversuch starten.

3. Schließlich kann es auch passieren, dass zwar alle $\mathcal{R}_i$ in der Folge terminieren, man aber kein konfluentes TES erreicht. Dann terminiert das Vervollständigungsverfahren nicht und in diesem Fall erkennen wir daher die Erfolglosigkeit des Verfahrens auch nicht.

Eine generelle Vermeidung der letzten beiden Möglichkeiten ist prinzipiell nicht möglich, da das Wortproblem im allgemeinen unentscheidbar ist. (Selbstverständlich kann man aber natürlich versuchen, bessere Vervollständigungsalgorithmen zu entwickeln, die häufiger erfolgreich sind, vgl. Abschnitt 6.2.) Man erhält demnach den folgenden Algorithmus BASIC_COMPLETION.

Algorithmus BASIC_COMPLETION$(\mathcal{E}, \succ)$
Eingabe: Ein Gleichungssystem $\mathcal{E}$ und eine Reduktionsordnung $\succ$.
Ausgabe: Ein zu $\mathcal{E}$ äquivalentes konvergentes TES $\mathcal{R}$
oder "*Fail*" oder Nicht-Terminierung.

1. Falls es $s \equiv t \ \in \mathcal{E}$ mit $s \neq t$, $s \not\succ t$ und $t \not\succ s$ gibt,
   dann gib "*Fail*" aus und brich ab.
2. Setze $i = 0$ und $\mathcal{R}_0 = \{l \to r | l \succ r \text{ und } l \equiv r \in \mathcal{E} \text{ oder } r \equiv l \in \mathcal{E}\}$.
3. Setze $\mathcal{R}_{i+1} = \mathcal{R}_i$.
4. Für alle $\langle s, t \rangle \in CP(\mathcal{R}_i)$:
   4.1. Berechne $\mathcal{R}_i$-Normalformen s' und t' von s und t.
   4.2. Falls $s' \neq t'$, $s' \not\succ t'$ und $t' \not\succ s'$,
   dann gib "*Fail*" aus und brich ab.
   4.3. Falls $s' \succ t'$, dann setze $\mathcal{R}_{i+1} = \mathcal{R}_{i+1} \cup \{s' \to t'\}$.
   4.4. Falls $t' \succ s'$, dann setze $\mathcal{R}_{i+1} = \mathcal{R}_{i+1} \cup \{t' \to s'\}$.
5. Falls $\mathcal{R}_{i+1} = \mathcal{R}_i$, dann gib $\mathcal{R}_i$ aus und brich ab.
6. Setze $i = i + 1$ und gehe zu Schritt 3.

In diesem Algorithmus gehen wir davon aus, dass die verwendete Reduktionsordnung $\succ$ bereits als Eingabe gegeben ist. (Wie erwähnt, sollte man bei der Implementierung dieses

Verfahrens die Präzedenz und den Status der RPOS solange wie möglich offen lassen und diese erst Schritt für Schritt festlegen.) In Schritt 1 und 2 werden die Gleichungen aus $\mathcal{E}$ so orientiert, dass die linken Seiten jeweils $\succ$-größer als die rechten Seiten sind. Triviale Gleichungen der Form $s = s$ werden hierbei weggelassen. Dies stellt sicher, dass das entstehende TES $\mathcal{R}_0$ terminiert.

In Schritt 4 werden für alle kritischen Paare $\langle s, t\rangle$ des aktuellen TES die entsprechenden Normalformen s' und t' berechnet. Falls $s' = t'$ gilt, so ist dieses kritische Paar zusammenführbar und damit unproblematisch. Ansonsten versucht man, die Gleichung $s' \equiv t'$ in eine Termersetzungsregel zu orientieren, deren Terminierung mit $\succ$ nachgewiesen werden kann. Falls dies gelingt, so wird die Regel dem bisherigen TES hinzugefügt. Auf diese Weise entsteht aus dem bisherigen TES $\mathcal{R}_i$ ein neues TES $\mathcal{R}_{i+1}$.

Aufgrund der Konstruktion der TESe ist die Terminierung aller $\mathcal{R}_i$ stets sicher gestellt, da sie nur Regeln $l \to r$ mit $l \succ r$ enthalten. Ebenso ist auch sicher gestellt, dass alle $\mathcal{R}_i$ zu $\mathcal{E}$ äquivalent sind. Die Adäquatheit folgt aus der Adäquatheit von $\mathcal{R}_0$ und daraus, dass $\mathcal{R}_0 \subseteq \mathcal{R}_i$ für alle i gilt. Die Korrektheit folgt, da kritische Paare stets $\mathcal{R}_i$-gleiche Terme in Beziehung setzen.

Falls in dem Schritt von $\mathcal{R}_i$ zu $\mathcal{R}_{i+1}$ keine neuen Regeln hinzugefügt werden, dann sind alle kritischen Paare von $\mathcal{R}_i$ zusammenführbar. Daher ist $\mathcal{R}_i$ lokal konfluent und aufgrund der Terminierung auch konfluent. In diesem Fall ist $\mathcal{R}_i$ also ein konvergentes zu $\mathcal{E}$ äquivalentes TES und dieses wird ausgegeben.

Ein Fehlschlag des Algorithmus entsteht, wenn neu hinzuzufügende Gleichungen mit der Reduktionsordnung $\succ$ nicht orientiert werden können. In diesem Fall könnte man den Vervollständigungsalgorithmus erneut mit einer anderen Reduktionsordnung $\succ'$ starten. Neben dem Erfolgsfall und dem Fehlschlag kann es auch passieren, dass der Algorithmus nicht terminiert. In diesem Fall werden insgesamt unendlich viele neue Regeln erzeugt.

Die folgenden Beispiele zeigen die drei verschiedenen Verhaltensweisen des Algorithmus.

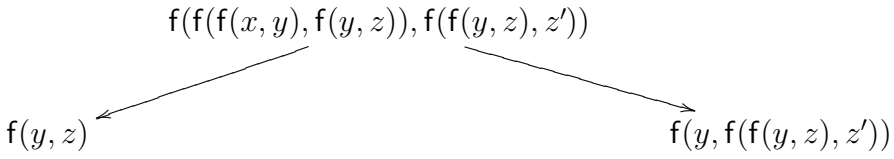
Beispiel 6.1.3 Wir betrachten eine Gleichung zur Axiomatisierung sogenannter *zentraler Gruppoide*.

$$\mathcal{E} = \{\mathsf{f}(\mathsf{f}(x, y), \mathsf{f}(y, z)) \equiv y\}$$

Wir rufen BASIC_COMPLETION mit $\mathcal{E}$ und einer (geeignet zu bestimmenden) Simplifikationsordnung $\succ$ auf. Bei jeder Simplifikationsordnung $\succ$ gilt $\mathsf{f}(\mathsf{f}(x, y), \mathsf{f}(y, z)) \succ y$ und somit ergibt sich

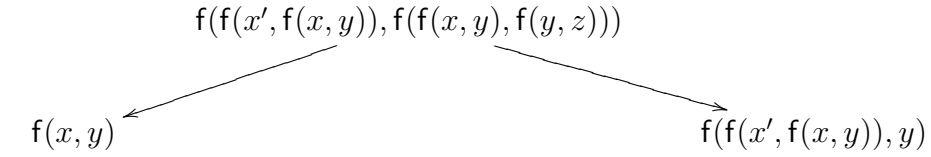
$$\mathcal{R}_0 = \{\mathsf{f}(\mathsf{f}(x, y), \mathsf{f}(y, z)) \to y\}.$$

Nun berechnen wir $\mathcal{R}_1$. Hierzu werden die kritischen Paare von $\mathcal{R}_0$ bestimmt, die sich durch Überlappung von $\mathsf{f}(\mathsf{f}(x, y), \mathsf{f}(y, z)) \to y$ mit seiner variablenumbenannten Kopie $\mathsf{f}(\mathsf{f}(x', y'), \mathsf{f}(y', z')) \to y'$ an den Stellen 1 und 2 ergibt. Die Überlappung an der Stelle 1 verwendet den mgu $\{x'/\mathsf{f}(x, y), y'/\mathsf{f}(y, z)\}$. Dies entspricht der kritischen Situation



bzw. dem kritischen Paar $\langle \mathsf{f}(y,z), \mathsf{f}(y,\mathsf{f}(\mathsf{f}(y,z),z')) \rangle$. Die beiden Terme des kritischen Paars sind bereits Normalformen und bei jeder Simplifikationsordnung $\succ$ gilt $\mathsf{f}(y,\mathsf{f}(\mathsf{f}(y,z),z')) \succ \mathsf{f}(y,z)$. Somit wird $\mathcal{R}_1$ um die Regel $\mathsf{f}(y,\mathsf{f}(\mathsf{f}(y,z),z')) \to \mathsf{f}(y,z)$ ergänzt.

Die Überlappung an der Stelle 2 verwendet den mgu $\{y'/\mathsf{f}(x,y), z'/\mathsf{f}(y,z)\}$. Dies entspricht der kritischen Situation



bzw. dem kritischen Paar $\langle \mathsf{f}(x,y), \mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),y) \rangle$. Die beiden Terme des kritischen Paars sind wieder Normalformen und bei jeder Simplifikationsordnung $\succ$ gilt $\mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),y) \succ \mathsf{f}(x,y)$. Somit wird $\mathcal{R}_1$ auch um die Regel $\mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),y) \to \mathsf{f}(x,y)$ ergänzt. So erhält man

$$\begin{array}{rrcl} \mathcal{R}_1 = & \{\mathsf{f}(\mathsf{f}(x,y),\mathsf{f}(y,z)) & \to & y, \\ & \mathsf{f}(y,\mathsf{f}(\mathsf{f}(y,z),z')) & \to & \mathsf{f}(y,z), \\ & \mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),y) & \to & \mathsf{f}(x,y)\}. \end{array}$$

In der nächsten Iteration stellt sich heraus, dass alle kritischen Paare von $\mathcal{R}_1$ zusammenführbar sind. Daher ergibt sich $\mathcal{R}_2 = \mathcal{R}_1$ und dieses TES wird als Ergebnis des Algorithmus ausgegeben. In der Tat ist es konvergent und äquivalent zu $\mathcal{E}$.

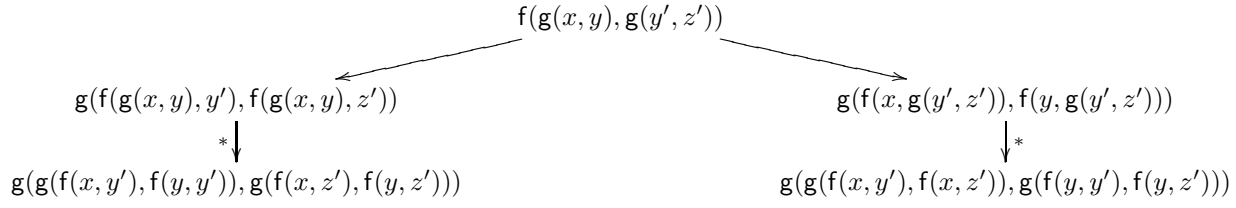
Beispiel 6.1.4 Wir betrachten ein Gleichungssystem, das ausdrückt, dass eine Funktion f links- und rechtsdistributiv über einer anderen Funktion g ist.

$$\mathcal{E} = \{\mathsf{f}(x,\mathsf{g}(y,z)) \equiv \mathsf{g}(\mathsf{f}(x,y),\mathsf{f}(x,z)), \mathsf{f}(\mathsf{g}(x,y),z) \equiv \mathsf{g}(\mathsf{f}(x,z),\mathsf{f}(y,z))\}.$$

Wir rufen wiederum BASIC_COMPLETION mit $\mathcal{E}$ und einer (geeignet zu bestimmenden) RPOS $\succ$ auf. Beispielsweise kann man die Gleichungen von links nach rechts orientieren, wenn man eine Präzedenz mit $\mathsf{f} \sqsupset \mathsf{g}$ wählt. So ergibt sich

$$\begin{array}{rrcl} \mathcal{R}_0 = & \{\mathsf{f}(x,\mathsf{g}(y,z)) & \to & \mathsf{g}(\mathsf{f}(x,y),\mathsf{f}(x,z)), \\ & \mathsf{f}(\mathsf{g}(x,y),z) & \to & \mathsf{g}(\mathsf{f}(x,z),\mathsf{f}(y,z))\}. \end{array}$$

Zur Bestimmung von $\mathcal{R}_1$ wird das einzige kritische Paar von $\mathcal{R}_0$ betrachtet, das sich durch Überlappung der beiden Regeln an der äußersten Position ergibt. So erhält man die folgende kritische Situation:



Im obigen Bild sind bereits die Normalformen der beiden Terme in dem entsprechenden kritischen Paar angegeben.

Da die Normalformen nicht identisch sind, müsste man nun eine Regel hinzufügen, die diese beiden Normalformen in Beziehung setzt. Leider sind diese beiden Terme jedoch bei keiner Reduktionsordnung vergleichbar. (In der Tat würde eine Regel zwischen diesen beiden Termen stets zur Nicht-Terminierung führen, denn falls die darin vorkommenden Variablen alle gleich instantiiert werden, würde die Regel einen Term in sich selbst überführen.) Das Vervollständigungsverfahren scheitert daher und der Algorithmus gibt "*Fail*" aus.

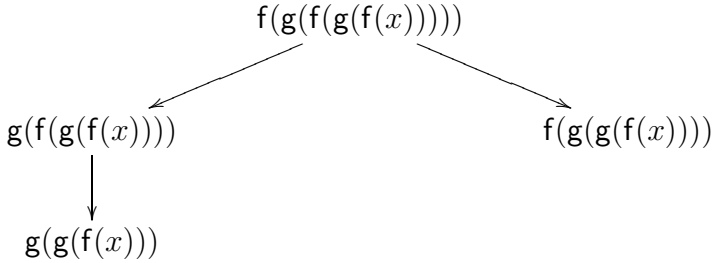
Beispiel 6.1.5 Um ein Beispiel für die Nicht-Terminierung des Vervollständigungsalgorithmus zu erhalten, betrachten wir die folgende Gleichung.

$$\mathcal{E} = \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \equiv \mathsf{g}(\mathsf{f}(x))\}$$

Zur Orientierung der Gleichung wählen wir wieder eine Simplifikationsordnung $\succ$. Daher kann die Gleichung von links nach rechts orientiert werden und man erhält

$$\mathcal{R}_0 = \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{g}(\mathsf{f}(x))\}.$$

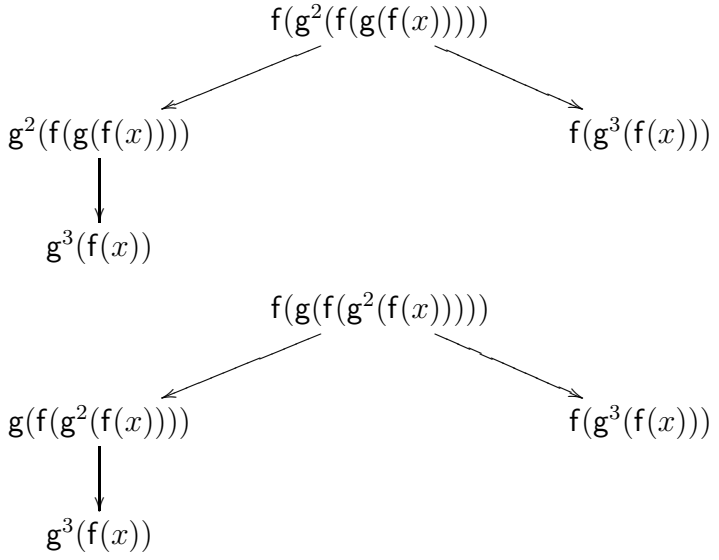
Nun bilden wir das einzige kritische Paar, das der folgenden kritischen Situation entspricht:



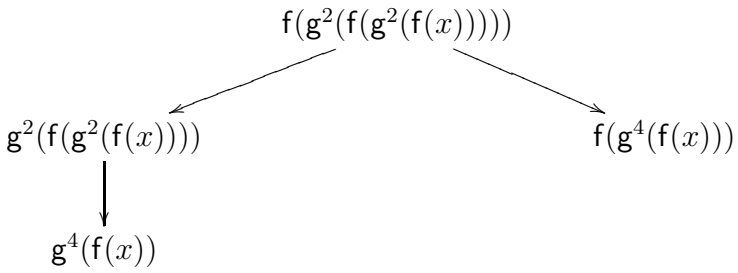
Dieses kritische Paar muss von rechts nach links orientiert werden, so dass sich

$$\begin{array}{rlcl} \mathcal{R}_1 = & \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) & \to & \mathsf{g}(\mathsf{f}(x)), \\ & \mathsf{f}(\mathsf{g}^2(\mathsf{f}(x))) & \to & \mathsf{g}^2(\mathsf{f}(x))\} \end{array}$$

ergibt. Nun berechnet man die kritischen Paare von $\mathcal{R}_1$. Das kritische Paar, das durch Überlagerung der ersten Regel mit sich selbst entsteht, ist nun zusammenführbar. Durch Überlagerung der ersten mit der zweiten Regel entstehen die beiden folgenden kritischen Situationen:



Durch Überlagerung der zweiten Regel mit sich selbst erhält man



Man erhält daher

$$\begin{array}{rlcl} \mathcal{R}_1 = & \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) & \to & \mathsf{g}(\mathsf{f}(x)), \\ & \mathsf{f}(\mathsf{g}^2(\mathsf{f}(x))) & \to & \mathsf{g}^2(\mathsf{f}(x)), \\ & \mathsf{f}(\mathsf{g}^3(\mathsf{f}(x))) & \to & \mathsf{g}^3(\mathsf{f}(x)), \\ & \mathsf{f}(\mathsf{g}^4(\mathsf{f}(x))) & \to & \mathsf{g}^4(\mathsf{f}(x))\} \end{array}$$

Dieses Vorgehen wird nun analog dazu unendlich oft wiederholt, so dass insgesamt eine unendliche Menge von Regeln

$$\mathcal{R}_\infty = \{\mathsf{f}(\mathsf{g}^n(\mathsf{f}(x))) \to \mathsf{g}^n(\mathsf{f}(x)) \mid n \geq 1\}$$

entsteht. In diesem Fall terminiert der Algorithmus BASIC_COMPLETION daher nicht.

Obwohl der Vervollständigungsalgorithmus im letzten Beispiel also nicht terminiert, so ist das entstehende *unendliche* TES $\mathcal{R}_\infty$ dennoch konvergent und äquivalent zu $\mathcal{E}$. In solchen Fällen kann man mit dem Vervollständigungsverfahren zwar kein Entscheidungsverfahren, aber ein *Semi-Entscheidungsverfahren* für das Wortproblem erhalten. Um $s \equiv_\mathcal{E} t$ zu untersuchen, betrachtet man zunächst das TES $\mathcal{R}_0$. Man untersucht nun, ob s und t in $\mathcal{R}_0$ zusammenführbar sind (dies ist entscheidbar, da $\mathcal{R}_0$ terminiert). In diesem Fall ist $s \equiv_\mathcal{E} t$ bewiesen. Ansonsten berechnet man die nächste Iteration $\mathcal{R}_1$ und wiederholt das Vorgehen, etc. Es ist sicher gestellt, dass es zu jeder Gleichung mit $s \equiv_\mathcal{E} t$ eine Iteration $\mathcal{R}_n$ gibt, so dass s und t in $\mathcal{R}_n$ (und in jedem $\mathcal{R}_m$ mit $m \geq n$) zusammenführbar sind. Insofern kann jede wahre Gleichung auf diese Art bewiesen werden und man hat damit ein Semi-Entscheidungsverfahren. Der folgende Satz zeigt die Korrektheit des Vervollständigungsalgorithmus.

Satz 6.1.6 (Korrektheit von BASIC_COMPLETION**)** *Sei $\mathcal{E}$ ein (endliches) Gleichungssystem und $\succ$ eine Reduktionsordnung.*

(a) Falls BASIC_COMPLETION$(\mathcal{E}, \succ)$ *terminiert und ein TES $\mathcal{R}_n$ als Resultat liefert, dann ist $\mathcal{R}_n$ ein (endliches) konvergentes TES, das äquivalent zu $\mathcal{E}$ ist. In diesem Fall lässt sich aus $\mathcal{R}_n$ mit dem Algorithmus* WORTPROBLEM *ein Entscheidungsverfahren für das Wortproblem über $\mathcal{E}$ erhalten.*

(b) Falls BASIC_COMPLETION$(\mathcal{E}, \succ)$ *nicht terminiert, dann ist $\mathcal{R}_\infty = \bigcup_{i \geq 0} \mathcal{R}_i$ ein unendliches konvergentes TES, das äquivalent zu $\mathcal{E}$ ist. In diesem Fall kann man den Vervollständigungsalgorithmus als Semi-Entscheidungsverfahren für das Wortproblem über $\mathcal{E}$ verwenden.*

Beweis.

(a) Für ein endliches TES ist die Menge der kritischen Paare stets endlich. In jeder Iteration wird das berechnete TES also nur um endlich viele Regeln erweitert. Da $\mathcal{R}_n$ nach endlich vielen Iterationen erzeugt wird, ist $\mathcal{R}_n$ endlich. Nach Konstruktion terminiert $\mathcal{R}_n$, da nach Konstruktion $l \succ r$ für alle Regeln $l \to r \in \mathcal{R}_n$ gilt. (Hieraus folgt die Terminierung nach Satz 4.3.4.) Die Konfluenz von $\mathcal{R}_n$ folgt, da der Algorithmus BASIC_COMPLETION nur dann $\mathcal{R}_n$ ausgibt, wenn alle seine kritischen Paare zusammenführbar sind. Aus dem Kritischen-Paar-Lemma (Satz 5.2.6) folgt die lokale Konfluenz und aus dem Diamond Lemma (Satz 5.2.3) folgt damit die Konfluenz von $\mathcal{R}_n$.

Zur Äquivalenz von $\mathcal{E}$ und $\mathcal{R}_n$ genügt es nach dem Satz von Birkhoff (Satz 3.1.14) zu zeigen, dass $\leftrightarrow^*_{\mathcal{R}_n} = \leftrightarrow^*_{\mathcal{E}}$ gilt. Die Adäquatheit von $\mathcal{R}_n$ folgt aus $\leftrightarrow^*_{\mathcal{R}_0} = \leftrightarrow^*_{\mathcal{E}}$ und aus $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \ldots \subseteq \mathcal{R}_n$. Damit ergibt sich $\leftrightarrow^*_{\mathcal{E}} = \leftrightarrow^*_{\mathcal{R}_0} \subseteq \leftrightarrow^*_{\mathcal{R}_n}$. Für die Korrektheit zeigen wir $\leftrightarrow^*_{\mathcal{R}_i} \subseteq \leftrightarrow^*_{\mathcal{E}}$ für alle i durch Induktion über i. Für $i = 0$ gilt dies nach Konstruktion. Für jedes kritische Paar $\langle s, t\rangle$ von $\mathcal{R}_i$ gilt $s \leftrightarrow^*_{\mathcal{R}_i} t$ und somit gilt auch für die entsprechenden $\mathcal{R}_i$-Normalformen s' und t', dass $s' \leftrightarrow^*_{\mathcal{R}_i} t'$ gilt. Somit folgt aus der Induktionshypothese $s' \leftrightarrow^*_{\mathcal{E}} t'$ und daher (aufgrund der Abgeschlossenheit von $\leftrightarrow^*_{\mathcal{E}}$ unter Substitutionen und Kontexten) $\leftrightarrow^*_{\mathcal{R}_{i+1}} \subseteq \leftrightarrow^*_{\mathcal{E}}$.

(b) Das System $\mathcal{R}_\infty$ ist unendlich, da in jeder Iteration des Algorithmus BASIC_COMPLETION mindestens eine Regel zu dem TES hinzugefügt wird. Dass $\mathcal{R}_\infty$ terminiert und zu $\mathcal{E}$ äquivalent ist, folgt wie in Teil (a).

Zur Konfluenz zeigen wir, dass alle kritischen Paare von $\mathcal{R}_\infty$ zusammenführbar sind. Das Kritische-Paar-Lemma und das Diamond Lemma (Satz 5.2.6 und 5.2.3) gelten auch bei unendlichen TESen, so dass daraus in der Tat die Konfluenz folgt. Sei $\langle s, t\rangle$ ein kritisches Paar von $\mathcal{R}_\infty$. Dann existieren Regeln $l_1 \to r_1$ und $l_2 \to r_2$ aus $\mathcal{R}_\infty$, deren Überlagerung das kritische Paar gebildet hat. Da $\mathcal{R}_\infty = \bigcup_{i \geq 0} \mathcal{R}_i$ und $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \mathcal{R}_2 \subseteq \ldots$ gilt, existiert ein $n \geq 0$ mit $l_1 \to r_1, l_2 \to r_2 \in \mathcal{R}_n$. Somit ist $\langle s, t\rangle \in CP(\mathcal{R}_n)$. Dann ist dieses kritische Paar aber entweder bereits in $\mathcal{R}_n$ oder spätestens in $\mathcal{R}_{n+1}$ zusammenführbar, da im zweiten Fall eine entsprechende Regel in $\mathcal{R}_{n+1}$ aufgenommen wurde. Damit ist $\langle s, t\rangle$ also auch in $\mathcal{R}_\infty \supseteq \mathcal{R}_{n+1}$ zusammenführbar. □

6.2 Ein verbesserter Vervollständigungsalgorithmus

Der Vervollständigungsalgorithmus BASIC_COMPLETION aus dem vorigen Abschnitt ist recht aufwendig, da er eine unnötig große Menge von Regeln erzeugt. In dem bisherigen Algorithmus gibt es keine Möglichkeit, Regeln, die man einmal erzeugt hat, später wieder zu löschen. Die große Menge an Regeln hat aber zur Folge, dass man alle diese Regeln bei der Berechnung kritischer Paare betrachten muss. Dies ist für einen praktischen Einsatz der Vervollständigungsprozedur ungünstig.

Aus diesem Grund sollte es während der Vervollständigung möglich sein, frühere Regeln mit den neu entstandenen Regeln zu vereinfachen. Falls sich frühere Regeln mit den

neuen Regeln so vereinfachen lassen, dass die linke und die rechte Seite zum gleichen Term reduziert werden können, so ist diese frühere Regel nicht mehr notwendig und man kann sie löschen.

Beispiel 6.2.1 Als Beispiel betrachten wir wieder unser Gruppen-TES. Als Reduktionsordnung bei der Vervollständigung verwenden wir wieder die RPOS, wobei das Funktionssymbol f den Status $\langle 2, 1\rangle$ hat. Unser Ausgangs-TES $\mathcal{R}_0$ bestand aus den folgenden Regeln.

$$\begin{array}{rcll} \mathsf{f}(x, \mathsf{f}(y, z)) & \to & \mathsf{f}(\mathsf{f}(x, y), z) & (G1) \\ \mathsf{f}(x, \mathsf{e}) & \to & x & (G2) \\ \mathsf{f}(x, \mathsf{i}(x)) & \to & \mathsf{e} & (G3) \end{array}$$

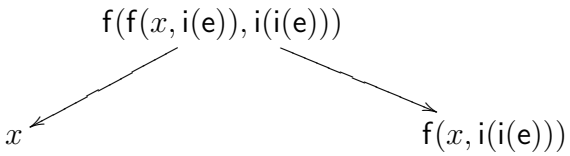
Aus $(G1)$ und $(G3)$ entstand das kritische Paar $\langle \mathsf{f}(\mathsf{f}(x, y), \mathsf{i}(y)), \mathsf{f}(x, \mathsf{e})\rangle$, das zu folgender neuen Regel führte:

$$\mathsf{f}(\mathsf{f}(x, y), \mathsf{i}(y)) \to x \quad (G4)$$

Nun muss man die neuen kritischen Paare betrachten, die sich aufgrund der Regel $(G4)$ ergeben. Aus $(G4)$ und $(G2)$ entsteht z.B. das kritische Paar $\langle x, \mathsf{f}(x, \mathsf{i}(\mathsf{e}))\rangle$, das zu der folgenden Regel führt:

$$\mathsf{f}(x, \mathsf{i}(\mathsf{e})) \to x \quad (G5)$$

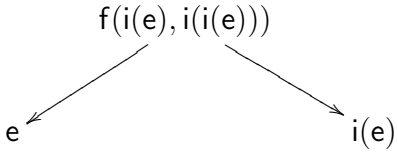
Im Verlauf der weiteren kritischen Paar-Bildung überlagert man auch die neuen Regeln $(G4)$ und $(G5)$, was die folgende kritische Situation



und damit die folgende neue Regel ergibt:

$$\mathsf{f}(x, \mathsf{i}(\mathsf{i}(\mathsf{e}))) \to x \quad (G6)$$

Die Überlagerung von $(G3)$ und $(G6)$ ergibt die kritische Situation



und die neue Regel:

$$\mathsf{i}(\mathsf{e}) \to \mathsf{e} \quad (G7)$$

An dieser Stelle erkennt man nun, dass die Regeln $(G5)$ und $(G6)$ nicht mehr notwendig sind, denn ihr Effekt kann vollständig mit der neuen Regel $(G7)$ (und der bereits existierenden Regel $(G2)$) erreicht werden. Genauer kann man mit Hilfe von $(G7)$ und $(G2)$ die linken und rechten Seiten der Regeln $(G5)$ und $(G6)$ zu dem gleichen Term wie die jeweilige rechte Seite reduzieren. Die damaligen kritischen Paare, zu deren Zusammenführbarkeit $(G5)$ und $(G6)$ nötig waren, können jetzt mit Hilfe von $(G7)$ zusammengeführt werden. Die Regeln $(G5)$ und $(G6)$ können daher aus dem TES gelöscht werden.

Unser Ziel ist nun, ein verbessertes Vervollständigungsverfahren zu entwickeln, bei dem ein solches Vereinfachen der Regeln untereinander (und ggf. das Löschen von Regeln) möglich ist. Das Vervollständigungsverfahren wird als eine Menge von Transformationsregeln angegeben, vgl. [Bac91]. Die Strategie, in welcher Reihenfolge oder mit welcher Priorität die Regeln anzuwenden sind, wird (fast) nicht festgelegt. Wir beschreiben auf diese Weise also eine ganze Menge von Vervollständigungsalgorithmen, die sich in dieser Strategie unterscheiden. Ein weiterer Vorteil des verbesserten Vervollständigungsverfahrens ist also, dass man nicht auf das Vorgehen in BASIC_COMPLETION festgelegt ist, sondern die Regeln in (nahezu) beliebiger Weise anwenden kann, so dass man oftmals wesentlich schneller ein konvergentes zu $\mathcal{E}$ äquivalentes TES findet.

Die Transformationsregeln arbeiten auf Paaren $(\mathcal{E}, \mathcal{R})$, wobei $\mathcal{E}$ ein (endliches) Gleichungssystem und $\mathcal{R}$ ein (ebenfalls endliches) TES ist. Die Transformation beginnt mit $\mathcal{R} = \varnothing$, wobei $\mathcal{E}$ das ursprüngliche Gleichungssystem ist, zu dem ein konvergentes äquivalentes TES gefunden werden soll. Wird $(\mathcal{E}, \mathcal{R})$ in $(\mathcal{E}', \mathcal{R}')$ überführt, so ist sicher gestellt, dass sich die dadurch beschriebene Gleichheitsrelation nicht ändert, d.h., es gilt $\leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'}$. Das Ziel der Transformation ist daher, das ursprüngliche Paar $(\mathcal{E}, \mathcal{R})$ mit $\mathcal{R} = \varnothing$ in mehreren Schritten zu einem Paar $(\mathcal{E}', \mathcal{R}')$ zu transformieren, bei dem $\mathcal{E}' = \varnothing$ ist. In diesem Fall gilt nämlich $\leftrightarrow^*_{\mathcal{E}} = \leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'} = \leftrightarrow^*_{\mathcal{R}'}$, d.h., $\mathcal{R}'$ ist ein zu $\mathcal{E}$ äquivalentes TES.

Genauer enthält die Gleichungsmenge $\mathcal{E}$ in einem Paar $(\mathcal{E}, \mathcal{R})$ im Verlauf der Transformation immer diejenigen Gleichungen, die noch nicht in Regeln überführt wurden und $\mathcal{R}$ ist ein terminierendes TES. Die Terminierung des TES wird wie im grundlegenden Vervollständigungsalgorithmus durch eine Reduktionsrelation $\succ$ sicher gestellt. $\mathcal{R}$ darf dann nur Regeln $l \to r$ enthalten, für die $l \succ r$ gilt.

Definition 6.2.2 (Transformationsregeln für Vervollständigung) *Sei $\mathcal{E}$ ein Gleichungssystem, $\mathcal{R}$ ein TES und $\succ$ eine Reduktionsordnung. Dann definieren wir die folgenden sechs Transformationsregeln auf Paaren $(\mathcal{E}, \mathcal{R})$.*

Generieren	$\dfrac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}$	*falls* $\langle s, t\rangle \in CP(\mathcal{R})$
Orientieren	$\dfrac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \to t\}}$	*falls* $s \succ t$
	$\dfrac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \to s\}}$	*falls* $t \succ s$
Löschen	$\dfrac{\mathcal{E} \cup \{s \equiv s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}}$	
Reduziere-Gleichung	$\dfrac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\mathcal{E} \cup \{u \equiv t\}, \mathcal{R}}$	*falls* $s \to_{\mathcal{R}} u$
	$\dfrac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\mathcal{E} \cup \{s \equiv v\}, \mathcal{R}}$	*falls* $t \to_{\mathcal{R}} v$

Reduziere-Rechts $\dfrac{\mathcal{E}, \mathcal{R} \cup \{s \to t\}}{\mathcal{E}, \mathcal{R} \cup \{s \to v\}}$ *falls* $t \to_{\mathcal{R}} v$

Reduziere-Links $\dfrac{\mathcal{E}, \mathcal{R} \cup \{s \to t\}}{\mathcal{E} \cup \{u \equiv t\}, \mathcal{R}}$ *falls* $s \to_{\mathcal{R}} u$ *mit Regel* $l \to r$ *gilt und* l *kann nicht mit* $s \to t$ *reduziert werden*

Wir schreiben $(\mathcal{E}, \mathcal{R}) \vdash_C (\mathcal{E}', \mathcal{R}')$*, falls* $(\mathcal{E}, \mathcal{R})$ *mit Hilfe einer dieser Transformationsregeln in* $(\mathcal{E}', \mathcal{R}')$ *überführt werden kann.*

Die Transformationsregel "Generieren" erzeugt eine neue Gleichung aus einem kritischen Paar von $\mathcal{R}$. Alle kritischen Paare lassen sich also als Gleichungen darstellen, die wieder zu neuen Regeln führen können (sofern die entsprechenden Terme nicht zusammenführbar sind).

Die Transformationsregel "Orientieren" überführt eine Gleichung in eine Regel, wobei sich die Orientierung aus der Reduktionsordnung $\succ$ ergibt. Dies entspricht dem Vorgehen in dem grundlegenden Vervollständigungsalgorithmus, bei dem ebenfalls die ursprünglichen Gleichungen (ebenso wie die aus kritischen Paaren entstehenden Gleichungen) auf diese Weise in Regeln überführt wurden. Die Transformationsregel "Löschen" löscht triviale Gleichungen aus $\mathcal{E}$, die in einem zu $\mathcal{E}$ äquivalenten TES nicht notwendig sind. Mit der Transformationsregel "Reduziere-Gleichung" können die Terme in Gleichungen vereinfacht werden, indem sie wiederholt mit den Regeln aus $\mathcal{R}$ reduziert werden. Indem man diese Transformationsregel stets mit oberster Priorität (ggf. nach der "Löschen"-Transformationsregel) so lange wie möglich anwendet, kann man in allen Gleichungen $s \equiv t$ aus $\mathcal{E}$ die Terme s und t zu ihren Normalformen reduzieren.

Mit den ersten vier Transformationsregeln kann man den grundlegenden Vervollständigungsalgorithmus aus dem vorigen Abschnitt nachbilden. Hierzu müssen diese Transformationsregeln mit folgender Strategie angewendet werden:

1. Wende zunächst "Löschen" solange wie möglich an.
2. Wende dann "Orientieren" solange wie möglich an.
3. Falls danach $\mathcal{E} \neq \varnothing$ ist, dann breche mit Fehlschlag ab.
4. Wende "Generieren" solange an, bis alle kritischen Paare von $\mathcal{R}$ erzeugt sind.
5. Wende "Reduziere-Gleichung" solange wie möglich an.
6. Wende dann "Löschen" solange wie möglich an.
7. Wenn nun $\mathcal{E} = \varnothing$ ist, dann breche ab und gib $\mathcal{R}$ aus.
8. Gehe zu Schritt 2.

Beispiel 6.2.3 Um die Arbeitsweise der Transformationsregeln zu illustrieren, betrachten wir noch einmal die Vervollständigung aus Bsp. 6.1.3 mit Hilfe des grundlegenden Vervollständigungsalgorithmus. Wenn man der obigen Strategie folgt, kann man diese Vervollständigung genauso auch mit den Transformationsregeln aus Def. 6.2.2 durchführen. Man beginnt mit

$$(\{\mathsf{f}(\mathsf{f}(x, y), \mathsf{f}(y, z)) \equiv y\}, \varnothing).$$

Da "Löschen" nicht anwendbar ist, wendet man nun "Orientieren" an.

$$(\varnothing, \{\mathsf{f}(\mathsf{f}(x, y), \mathsf{f}(y, z)) \to y\})$$

Nun wenden wir "Generieren" zweimal an, um die beiden kritischen Paare zu erzeugen.

$$(\{\mathsf{f}(y,z) \equiv \mathsf{f}(y,\mathsf{f}(\mathsf{f}(y,z),z')), \mathsf{f}(x,y) \equiv \mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),y)\}, \{\mathsf{f}(\mathsf{f}(x,y),\mathsf{f}(y,z)) \to y\})$$

Weder "Reduziere-Gleichung" noch "Löschen" sind anwendbar. Somit springt man in der Strategie zu Schritt 2 zurück und orientiert die beiden Paare.

$$(\varnothing, \{\mathsf{f}(\mathsf{f}(x,y),\mathsf{f}(y,z)) \to y, \mathsf{f}(y,\mathsf{f}(\mathsf{f}(y,z),z')) \to \mathsf{f}(y,z), \mathsf{f}(\mathsf{f}(x',\mathsf{f}(x,y)),y) \to \mathsf{f}(x,y)\})$$

Nun wendet man "Generieren" an und reduziert dann die aus den kritischen Paaren entstandenen Gleichungen mit "Reduziere-Gleichung". Hierbei lassen sich alle Gleichungen zu trivialen Gleichungen reduzieren, die dann mit "Löschen" eliminiert werden. In Schritt 6 ist daher $\mathcal{E} = \varnothing$ und damit wird das aus den drei obigen Regeln bestehende TES als Ergebnis ausgegeben.

Die Transformationsregeln "Reduziere-Rechts" und "Reduziere-Links" sind Verbesserungen gegenüber dem grundlegenden Vervollständigungsalgorithmus. "Reduziere-Rechts" erlaubt es, jederzeit die rechten Seiten von Regeln weiter auszuwerten. Dies ist insbesondere dann nützlich, wenn im Verlauf der Vervollständigung neue Regeln entstanden sind, die die ursprünglichen Regeln vereinfachen können. Die Terminierung von $\mathcal{R}$ wird durch diese Transformation nicht zerstört, denn nach Voraussetzung gilt $s \succ t$ und (aufgrund der Monotonie und Stabilität von $\succ$) $t \succ v$. Aus der Transitivität von $\succ$ folgt also $s \succ v$, d.h., die Terminierung des neuen TES lässt sich nach wie vor mit der Reduktionsordnung $\succ$ zeigen.

Im Unterschied hierzu kann das Reduzieren auf der linken Seite von Regeln die Terminierung zerstören. Wenn die linke Seite s einer Regel $s \to t$ zu u reduziert werden kann, so ist nicht sicher gestellt, ob $u \succ t$ gilt. Beispielsweise lässt sich durch wiederholte Reduktion mit den übrigen Regeln die linke Seite der Regel

$$\mathsf{f}(x,\mathsf{i}(\mathsf{i}(\mathsf{e}))) \quad \to \quad x \quad (G6)$$

zu x reduzieren, vgl. Bsp. 6.2.1. Diese neue linke Seite ist aber nicht mehr größer als die rechte Seite, d.h., es gilt $x \not\succ x$. Aus diesem Grund muss eine derartig reduzierte Regel wieder zurück in die Menge der Gleichheiten verschoben werden, denn es muss erneut überprüft werden, ob und ggf. wie man diese Regel orientieren kann.

Darüber hinaus ist die Reduktion von linken Regelseiten s nur mit solchen Regeln $l \to r$ erlaubt, bei denen s keinen Teilterm von l matcht, d.h., l lässt sich mit der Regel $s \to t$ nicht reduzieren. Der Grund für diese Einschränkung wird in Bsp. 6.2.12 deutlich werden. Falls $\mathcal{R} = \{\mathsf{f}(x,x) \to x, \mathsf{f}(x,y) \to x\}$ ist, so kann man also mit der zweiten Regel die erste Regel simplifizieren, denn umgekehrt wäre die erste Regel nicht zur Reduktion der linken Seite der zweiten Regel verwendbar. Auf diese Weise wird die erste Regel zur Gleichung $x \equiv x$ reduziert, die anschließend mit der "Löschen"-Transformationsregel gelöscht werden könnte. Hingegen darf man bei $\mathcal{R} = \{\mathsf{f}(x,y) \to x, \mathsf{f}(x,y) \to y\}$ diese Transformationsregel nicht anwenden.

Beispiel 6.2.4 Das folgende Beispiel zeigt, dass man durch die neuen Transformationsregeln TESe vervollständigen kann, bei denen der grundlegende Vervollständigungsalgorithmus fehlschlägt. Hierzu betrachten wir das folgende Gleichungssystem.

$$\mathcal{E} = \{\mathsf{h}(x,y) \equiv \mathsf{f}(x), \mathsf{h}(x,y) \equiv \mathsf{f}(y), \mathsf{g}(x,y) \equiv \mathsf{h}(x,y), \mathsf{g}(x,y) \equiv \mathsf{a}\}$$

Wir beginnen also mit dem Paar $(\mathcal{E}, \varnothing)$. Wenn wir die Transformationsregeln nach der Strategie des grundlegenden Vervollständigungsalgorithmus anwenden, so werden zunächst alle Regeln orientiert. Wenn man die RPOS mit der Präzedenz $\mathsf{g} \sqsupset \mathsf{h} \sqsupset \mathsf{f} \sqsupset \mathsf{a}$ verwendet, so erhält man schließlich das Paar $(\varnothing, \mathcal{R})$ mit

$$\mathcal{R} = \{\mathsf{h}(x,y) \to \mathsf{f}(x), \mathsf{h}(x,y) \to \mathsf{f}(y), \mathsf{g}(x,y) \to \mathsf{h}(x,y), \mathsf{g}(x,y) \to \mathsf{a}\}.$$

Mit Hilfe der Transformationsregel "Generieren" erzeugt man nun die beiden Gleichungen $\mathsf{f}(x) \equiv \mathsf{f}(y)$ und $\mathsf{h}(x,y) \equiv \mathsf{a}$ aus den kritischen Paaren. Das erste kritische Paar lässt sich aber nicht zu einer Regel orientieren, denn für keine Reduktionsordnung gilt $\mathsf{f}(x) \succ \mathsf{f}(y)$ oder $\mathsf{f}(y) \succ \mathsf{f}(x)$. Daher scheitert hier der grundlegende Vervollständigungsalgorithmus.

Bei (bislang) nicht orientierbaren Gleichungen muss man aber eigentlich die Vervollständigung noch nicht abbrechen, sondern man kann die Behandlung dieser Gleichung erst einmal zurückstellen. Dies führt in unserem Beispiel zum Erfolg. Mit der Transformationsregel "Reduziere-Gleichung" wird $\mathsf{h}(x,y) \equiv \mathsf{a}$ zu $\mathsf{f}(x) \equiv \mathsf{a}$ oder zu $\mathsf{f}(y) \equiv \mathsf{a}$ reduziert. Anschließend wird diese Gleichung von links nach rechts orientiert und als neue Regel aufgenommen. Man erhält so das Paar $(\{\mathsf{f}(x) \equiv \mathsf{f}(y)\}, \mathcal{R} \cup \{\mathsf{f}(x) \to \mathsf{a}\})$. Nun kann man mit dieser neuen Regel durch die "Reduziere-Gleichung"-Technik die Terme $\mathsf{f}(x)$ und $\mathsf{f}(y)$ in der Gleichung $\mathsf{f}(x) \equiv \mathsf{f}(y)$ beide zu a reduzieren. Die entstehende Gleichung kann dann mit der "Löschen"-Transformationsregel eliminiert werden. So ergibt sich das Paar $(\varnothing, \mathcal{R} \cup \{\mathsf{f}(x) \to \mathsf{a}\})$.

Das TES $\mathcal{R} \cup \{\mathsf{f}(x) \to \mathsf{a}\}$ kann nun sogar mit der Transformationsregel "Reduziere-Rechts" noch vereinfacht werden, da man die rechte Seite $\mathsf{h}(x,y)$ der dritten Regel mit der ersten oder der zweiten Regel zu $\mathsf{f}(x)$ oder $\mathsf{f}(y)$ reduzieren kann. Anschließend kann dieser Term weiter zu a reduziert werden. Ebenso können auch die rechten Seiten der ersten und zweiten Regel zu a reduziert werden. So ergibt sich schließlich das folgende konvergente und zu $\mathcal{E}$ äquivalente TES:

$$\{\mathsf{h}(x,y) \to \mathsf{a}, \mathsf{g}(x,y) \to \mathsf{a}, \mathsf{f}(x) \to \mathsf{a}\}.$$

Um die Korrektheit der Transformationsregeln bzw. des damit entstehenden Vervollständigungsverfahrens zu zeigen, beweisen wir zunächst, dass die dabei entstehenden TESe stets terminieren. Wenn die Terminierung von $\mathcal{R}$ mit der Reduktionsordnung $\succ$ gezeigt werden konnte, so gilt dies auch für das TES $\mathcal{R}'$, falls $(\mathcal{E}, \mathcal{R}) \vdash_C (\mathcal{E}', \mathcal{R}')$ gilt. Falls man also mit einem Paar $(\mathcal{E}, \varnothing)$ beginnt und daraus (in mehreren Schritten) $(\mathcal{E}', \mathcal{R}')$ herleitet, so folgt damit die Terminierung von $\mathcal{R}'$.

Lemma 6.2.5 (Terminierung des vervollständigten TES) *Sei $\succ$ die bei der Transformation aus Def. 6.2.2 verwendete Reduktionsordnung. Falls $l \succ r$ für alle $l \to r \in \mathcal{R}$ gilt und $(\mathcal{E}, \mathcal{R}) \vdash_C (\mathcal{E}', \mathcal{R}')$, so gilt auch $l \succ r$ für alle $l \to r \in \mathcal{R}'$.*

Beweis. In den Transformationsregeln "Generieren", "Löschen" und "Reduziere-Gleichung" wird $\mathcal{R}$ nicht verändert und in der Transformationsregel "Reduziere-Links" wird $\mathcal{R}$ ggf. um

eine Regel verkleinert. Bei diesen Transformationsregeln ist die Behauptung daher offensichtlich.

In der Transformationsregel "Orientieren" wird $\mathcal{R}$ um eine Regel $l \to r$ erweitert, für die nach Konstruktion $l \succ r$ gilt.

In der Transformationsregel "Reduziere-Rechts" folgt die Behauptung, da $t \to_{\mathcal{R}} v$ bedeutet, dass $t \succ v$ gilt (da in allen Regeln von $\mathcal{R}$ die linke Seite $\succ$-größer als die rechte ist und da $\succ$ unter Substitutionen und Kontext abgeschlossen ist). Somit ergibt sich $s \succ t \succ v$ und damit $s \succ v$ aufgrund der Transitivität von $\succ$. □

Das nächste Lemma zeigt, dass die Transformationsregeln "korrekt" sind, d.h., die Menge der gültigen Gleichungen wird dadurch nicht verändert.

Lemma 6.2.6 (Korrektheit des vervollständigten TES) *Aus* $(\mathcal{E}, \mathcal{R}) \vdash_C (\mathcal{E}', \mathcal{R}')$ *folgt* $\leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'}$.

Beweis. Für die "Generieren"-Transformationsregel gilt die Aussage, da $s \leftrightarrow^*_{\mathcal{R}} t$ für alle $\langle s, t\rangle \in CP(\mathcal{R})$ gilt. Für die Transformationsregeln "Orientieren" und "Löschen" ist die Aussage trivial.

Für die Transformationsregel "Reduziere-Gleichung" ist zu zeigen, dass $\leftrightarrow^*_{\mathcal{E}\cup\{s\equiv t\}\cup\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}\cup\{u\equiv t\}\cup\mathcal{R}}$ gilt. Die "$\supseteq$"-Richtung gilt, denn wir haben $u \leftarrow_{\mathcal{R}} s \to_{\mathcal{E}\cup\{s\equiv t\}} t$ und somit $u \leftrightarrow^*_{\mathcal{E}\cup\{s\equiv t\}\cup\mathcal{R}} t$ (aufgrund der Abgeschlossenheit von $\leftrightarrow^*_{\mathcal{E}\cup\{s\equiv t\}\cup\mathcal{R}}$ unter Substitutionen und Kontexten). Für die "$\subseteq$"-Richtung zeigt man $s \to_{\mathcal{R}} u \to_{\mathcal{E}\cup\{u\equiv t\}} t$ und somit $s \leftrightarrow^*_{\mathcal{E}\cup\{u\equiv t\}\cup\mathcal{R}} t$. Die Korrektheit der Version der "Reduziere-Gleichung"-Transformationsregel bei Reduktion auf rechten Seiten von Gleichungen sowie der Transformationsregeln "Reduziere-Rechts" und "Reduziere-Links" zeigt man auf analoge Weise. □

Aus den beiden obigen Lemmata erhält man die folgende Korrektheitsaussage: Falls man aus $(\mathcal{E}, \varnothing)$ in endlich vielen Schritten ein Paar $(\varnothing, \mathcal{R})$ erhält, bei dem alle kritischen Paare von $\mathcal{R}$ zusammenführbar sind, dann ist $\mathcal{R}$ ein konvergentes und zu $\mathcal{E}$ äquivalentes TES. Diese Aussagen lassen sich aber noch verbessern. Zum einen braucht man einmal untersuchte kritische Paare nicht noch einmal betrachten und zum anderen kann man eine entsprechende Aussage auch für unendliche Transformationsfolgen treffen. Um die Ergebnisse von endlichen und unendlichen Transformationen einheitlich beschreiben zu können, werden die folgenden Begriffe eingeführt.

Definition 6.2.7 (Persistente Gleichungen und Regeln)

- *Für eine endliche Folge von Transformationsschritten* $(\mathcal{E}_0, \mathcal{R}_0) \vdash_C (\mathcal{E}_1, \mathcal{R}_1) \vdash_C \ldots \vdash_C (\mathcal{E}_n, \mathcal{R}_n)$ *definieren wir die* persistenten Gleichungen $\mathcal{E}_\omega = \mathcal{E}_n$ *und die* persistenten Regeln $\mathcal{R}_\omega = \mathcal{R}_n$.

- *Für eine unendliche Folge von Transformationsschritten* $(\mathcal{E}_0, \mathcal{R}_0) \vdash_C (\mathcal{E}_1, \mathcal{R}_1) \vdash_C \ldots$ *definieren wir die* persistenten Gleichungen $\mathcal{E}_\omega = \bigcup_{i\geq 0}\bigcap_{j\geq i} \mathcal{E}_j$ *und die* persistenten Regeln $\mathcal{R}_\omega = \bigcup_{i\geq 0}\bigcap_{j\geq i} \mathcal{R}_j$.

Die persistenten Gleichungen bzw. Regeln sind also diejenigen Gleichungen bzw. Regeln, die im Verlauf der Transformation irgendwann eingeführt und nie wieder eliminiert werden.

In der grundlegenden Vervollständigungsprozedur (bzw. bei Anwendung der Transformationsregeln "Generieren", "Orientieren", "Löschen" und "Reduziere-Gleichung" nach der dementsprechenden Strategie) ergeben sich folgende persistente Gleichungen und Regeln: Wenn die Prozedur mit Erfolg terminiert, dann entspricht dies einer (endlichen) Transformationsfolge mit $\mathcal{E}_\omega = \varnothing$ und $\mathcal{R}_\omega$ ist dann ein endliches, konvergentes und zu dem ursprünglichen Gleichungssystem äquivalentes TES. Wenn die Prozedur nicht terminiert, so haben wir eine unendliche Transformationsfolge, bei der ebenfalls $\mathcal{E}_\omega = \varnothing$ gilt (denn jede irgendwann entstehende Gleichung wird entweder gelöscht oder zu einer neuen Regel umgewandelt) und $\mathcal{R}_\omega$ ist ein unendliches, konvergentes und zu dem ursprünglichen Gleichungssystem äquivalentes TES (vgl. Satz 6.1.6). Falls die Prozedur zum Fehlschlag führt, so haben wir eine endliche Transformationsfolge mit $\mathcal{E}_\omega \neq \varnothing$. Allgemein erhält man die folgende Definition für beliebige Transformationsfolgen und Vervollständigungsprozeduren.

Definition 6.2.8 (Erfolg und Korrektheit von Vervollständigungsprozeduren)
Eine (endliche oder unendliche) Transformationsfolge $(\mathcal{E}_0, \varnothing) \vdash_C (\mathcal{E}_1, \mathcal{R}_1) \vdash_C \ldots$ *ist* erfolgreich *gdw.* $\mathcal{E}_\omega = \varnothing$ *gilt und* $\mathcal{R}_\omega$ *ein konvergentes zu* $\mathcal{E}_0$ *äquivalentes TES ist. Eine Transformationsfolge* schlägt fehl *gdw.* $\mathcal{E}_\omega \neq \varnothing$ *ist. Eine* Vervollständigungsprozedur *ist ein Algorithmus, der als Eingabe ein Gleichungssystem* $\mathcal{E}_0$ *und eine Reduktionsordnung* $\succ$ *bekommt und eine (endliche oder unendliche) Transformationsfolge* $(\mathcal{E}_0, \varnothing) \vdash_C (\mathcal{E}_1, \mathcal{R}_1) \vdash_C \ldots$ *mit den Transformationsregeln aus Def. 6.2.2 und der Reduktionsordnung* $\succ$ *berechnet. Eine Vervollständigungsprozedur ist* korrekt *wenn für alle* $\mathcal{E}_0$ *und* $\succ$ *die berechnete Transformationsfolge entweder erfolgreich ist oder fehlschlägt.*

Aus den Transformationsregeln lassen sich also sofort beliebige Vervollständigungsprozeduren erhalten. Diese wenden (nach bestimmten Strategien) die Transformationsregeln an, wobei man mit der eingegeben Gleichungsmenge $\mathcal{E}_0$ und dem leeren TES beginnt und die eingegebene Reduktionsordnung $\succ$ verwendet. Die Vervollständigungsprozedur kann nach bestimmten Kriterien abbrechen (auch wenn noch Transformationsregeln anwendbar sind) oder unendlich laufen. Unabhängig davon, ob die Prozedur abbricht oder unendlich lange läuft, sollte allerdings folgendes gelten: Wenn die Menge der persistenten Gleichungen leer ist, dann sollten die persistenten Regeln ein zu $\mathcal{E}_0$ äquivalentes und konvergentes TES sein. Nur dann bezeichnet man eine Vervollständigungsprozedur als "korrekt". Satz 6.1.6 zeigt, dass die grundlegende Vervollständigungsprozedur korrekt im Sinne von Def. 6.2.8 ist. Wir werden im folgenden ein generelles Kriterium kennen lernen, um die Korrektheit von Vervollständigungsprozeduren sicher zu stellen.

Falls eine korrekte Vervollständigungsprozedur leere persistente Gleichungen erzeugt und endlich läuft, so kann man dann also die persistenten Regeln als Ergebnis zurückgeben und falls die Transformationsregeln unendlich oft angewendet werden, so lässt sich die Vervollständigungsprozedur immerhin als Semi-Entscheidungsalgorithmus verwenden. Falls die Menge der persistenten Gleichungen nicht leer ist, so ist die Vervollständigung gescheitert.

Die grundlegende Vervollständigungsprozedur bricht mit Fehlschlag ab, sobald eine Gleichung nicht (nach Reduktion) gelöscht oder in eine Regel orientiert werden kann. Falls die Menge der persistenten Gleichungen nicht leer ist, so stoppt sie also stets nach endlicher Zeit mit Fehlschlag. Im allgemeinen kann man aber auch unendliche fehlschlagende Transformationsfolgen haben.

Beispiel 6.2.9 Wir betrachten die Gleichungsmenge

$$\mathcal{E} = \{\mathsf{h}(x,y) \equiv \mathsf{f}(x), \mathsf{h}(x,y) \equiv \mathsf{f}(y), \mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \equiv \mathsf{g}(\mathsf{f}(x))\}.$$

Diese entsteht durch Kombination von Gleichungen aus den Beispielen 6.1.5 und 6.2.4. Die zu Regeln gerichteten Gleichungen $\mathsf{h}(x,y) \to \mathsf{f}(x)$ und $\mathsf{h}(x,y) \to \mathsf{f}(y)$ führen zu einem kritischen Paar, das die nicht-orientierbare Gleichung $\mathsf{f}(x) \equiv \mathsf{f}(y)$ ergibt. Während in Bsp. 6.2.4 diese Gleichung später reduziert werden konnte, entsteht in diesem Beispiel keine neue Regel, die diese Terme vereinfachen könnte. Stattdessen führt die Gleichung $\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \equiv \mathsf{g}(\mathsf{f}(x))$ wie in Bsp. 6.1.5 zu einer Nicht-Terminierung der Vervollständigung. So entsteht eine unendliche Transformationsfolge mit der persistenten Gleichung $\mathcal{E}_\omega = \{\mathsf{f}(x) \equiv \mathsf{f}(y)\}$ und den persistenten Regeln $\mathcal{R}_\omega = \{\mathsf{f}(\mathsf{g}^n(\mathsf{f}(x))) \to \mathsf{g}^n(\mathsf{f}(x)) \mid n \geq 1\}$.

Ein Vervollständigungsverfahren ist korrekt, wenn im Fall der leeren persistenten Gleichungen die persistenten Regeln $\mathcal{R}_\omega$ stets ein konvergentes und zur ursprünglichen Gleichungsmenge äquivalentes TES sind. Die Terminierung von $\mathcal{R}_\omega$ ist dabei durch die Transformationsregeln sicher gestellt, aber die Konfluenz von $\mathcal{R}_\omega$ gilt nicht ohne weitere Voraussetzung. Man muss voraussetzen, dass jedes kritische Paar der persistenten Regeln auch tatsächlich irgendwann einmal gebildet wurde. Sonst könnte man z.B. die Gleichung $\{\mathsf{a} \equiv \mathsf{b}, \mathsf{a} \equiv \mathsf{c}\}$ in das nicht-konfluente System $\{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{c}\}$ überführen und dann die Vervollständigung abbrechen. Ebenso könnte man in Bsp. 6.2.9 eine unendliche Transformationsfolge erhalten, die niemals das kritische Paar aus $\mathsf{h}(x,y) \to \mathsf{f}(x)$ und $\mathsf{h}(x,y) \to \mathsf{f}(y)$ bildet. Die persistenten Gleichungen wären dann leer, aber das (unendliche) TES $\mathcal{R}_\omega$ würde diese beiden Regeln enthalten, die die Konfluenz zerstören. Man muss sich also auf sogenannte *faire* Transformationsfolgen beschränken.

Definition 6.2.10 (Fairness) *Eine (endliche oder unendliche) Transformationsfolge* $(\mathcal{E}_0, \mathcal{R}_0) \vdash_C (\mathcal{E}_1, \mathcal{R}_1) \vdash_C \ldots$ *ist* fair*, falls* $CP(\mathcal{R}_\omega) \subseteq \bigcup_{i \geq 0} \mathcal{E}_i$*, d.h., jedes kritische Paar der persistenten Regeln trat auch einmal als Gleichung auf.*

Das grundlegende Theorem zur Vervollständigung zeigt nun, dass dies die einzige Einschränkung an die Strategie ist, die man für korrekte Vervollständigungsprozeduren braucht. Der Beweis dieses Satzes ist nicht trivial; man benötigt hierzu das Konzept der *Beweisordnungen*. Hierzu wird auf [BN98, Abschnitt 7.3] verwiesen.

Satz 6.2.11 (Vervollständigungssatz) *Jede Vervollständigungsprozedur, bei der jede nicht-fehlschlagende Transformationsfolge fair ist, ist korrekt.*

Der obige Satz gilt nur, wenn man die Transformationsregel "Reduziere-Links" wie in Def. 6.2.2 so einschränkt, dass eine Regel $s \to t$ nur mit einer anderen Regel $l \to r$ reduziert werden darf, bei der l selbst nicht mit $s \to t$ reduziert werden kann. Der Grund für diese Einschränkung bei der Transformationsregel "Reduziere-Links" ist, dass man ansonsten faire Transformationsfolgen konstruieren kann, bei denen die entstehenden persistenten Regeln nicht äquivalent zu $\mathcal{E}$ sind.

Beispiel 6.2.12 Wir betrachten die Gleichungsmenge

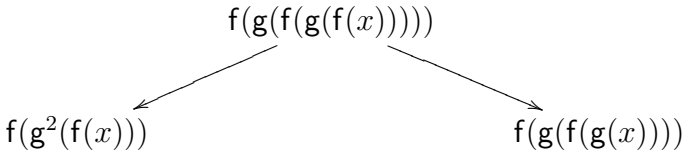
$$\mathcal{E} = \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \equiv \mathsf{f}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(x)) \equiv \mathsf{g}(x)\}.$$

Wir werden nun eine Transformationsfolge konstruieren, bei der unendlich viele Regeln $l_n \to r_n$ mit identischer linker Seite $l_n = \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$, aber verschiedener rechter Seite $r_n = \mathsf{f}(\mathsf{g}^{2^n}(x))$ entstehen. Wenn man die Anwendungsbedingung bei der Transformationsregel "Reduziere-Links" außer acht lässt, dann kann man für alle $n \geq 1$ die Regel $l_n \to r_n$ mit der Regel $l_{n+1} \to r_{n+1}$ reduzieren. Daher ist keine der Regeln $l_n \to r_n$ mit $n \geq 1$ persistent.

Durch zweifache Anwendung der "Orientieren"-Transformationsregel erhält man das Paar

$$(\varnothing, \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}(x)), \mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)\}).$$

Die Überlappung der zweiten Regel mit sich selbst ergibt ein kritisches Paar, das zusammengeführt werden kann. Somit wird dadurch keine neue Regel erzeugt. Aus der ersten Regel entsteht die kritische Situation



Die entstehende Gleichung $\mathsf{f}(\mathsf{g}^2(\mathsf{f}(x))) \equiv \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(x))))$ kann durch "Reduziere-Gleichung" zu $\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \equiv \mathsf{f}(\mathsf{g}^2(x))$ reduziert werden. Hierbei könnte man natürlich "Reduziere-Gleichung" noch weiter anwenden; dies ist jedoch durch die Einschränkung auf faire Transformationsfolgen nicht verlangt. Bei Verwendung der RPOS mit der Präzedenz $\mathsf{f} \sqsupset \mathsf{g}$ wird dies in eine Regel von links nach rechts orientiert. So erhält man

$$(\varnothing, \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}(x)), \mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}^2(x)), \mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)\}).$$

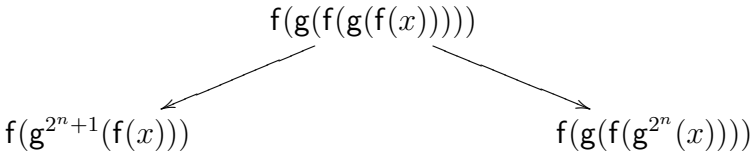
Wenn man die Anwendungsbedingung von "Reduziere-Links" ignoriert, kann man nun die erste Regel mit Hilfe der zweiten Regel zu der Gleichung $\mathsf{f}(\mathsf{g}^2(x)) \equiv \mathsf{f}(\mathsf{g}(x))$ reduzieren, die mit Hilfe der Regel $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)$ zu einer trivialen Gleichung reduziert und anschließend gelöscht wird. Wir haben daher nun das folgende Paar erzeugt:

$$(\varnothing, \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}^2(x)), \mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)\}).$$

Diese Konstruktion wird nun unendlich oft wiederholt. Wenn wir bereits das Paar

$$(\varnothing, \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}^{2^n}(x)), \mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)\})$$

haben, so bilden wir die kritische Situation



Der Term $\mathsf{f}(\mathsf{g}^{2^n+1}(\mathsf{f}(x)))$ lässt sich durch mehrfache Reduktion mit der Regel $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)$ zu $\mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$ reduzieren. Der Term $\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}^{2^n}(x))))$ wird durch die erste Regel zu $\mathsf{f}(\mathsf{g}^{2^n}(\mathsf{g}^{2^n}(x))) = \mathsf{f}(\mathsf{g}^{2^{n+1}}(x))$ reduziert. Die entstehende Gleichung wird von links nach rechts orientiert, was

$$(\varnothing, \{\mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}^{2^n}(x)), \mathsf{f}(\mathsf{g}(\mathsf{f}(x))) \to \mathsf{f}(\mathsf{g}^{2^{n+1}}(x)), \mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)\})$$

ergibt. Falls wir wieder die Anwendungsbedingung von "Reduziere-Links" ignorieren, so kann man die erste Regel mit Hilfe der zweiten Regel in die Gleichung $\mathsf{f}(\mathsf{g}^{2^{n+1}}(x)) \equiv \mathsf{f}(\mathsf{g}^{2^n}(x))$ überführen, die mit der Regel $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)$ in eine triviale Gleichung überführt und anschließend gelöscht wird.

Auf diese Weise entsteht eine unendliche, faire Transformationsfolge mit leerer Menge von persistenten Gleichungen. Die einzige persistente Regel ist $\mathcal{R}_\omega = \{\mathsf{g}(\mathsf{g}(x)) \to \mathsf{g}(x)\}$. Aber dieses TES ist nicht äquivalent zur ursprünglichen Gleichungsmenge $\mathcal{E}$.

Die grundlegende Vervollständigungsprozedur bildet alle kritischen Paare der entstandenen Regeln. Jede seiner Transformationsfolgen ist daher fair und nach Satz 6.2.11 ist diese Prozedur daher korrekt im Sinne von Def. 6.2.8. Dies bestätigt also unseren Korrektheitssatz 6.1.6.

Selbstverständlich stellt man an eine gute Vervollständigungsprozedur neben der Korrektheit den Anspruch, möglichst effizient zu sein und gleichzeitig möglichst selten fehlzuschlagen. Hierzu existieren verschiedene Strategien. Generell ist es vorteilhaft, die "Löschen"-Transformationsregel mit höchster Präferenz und die letzten drei Transformationsregeln mit nächsthöherer Präferenz anzuwenden, da sie die betrachteten Gleichungen und Regeln vereinfachen können. Die Transformationsregel "Orientieren" liegt in der Präferenz darunter und die Transformationsregel "Generieren" sollte man nur dann anwenden, wenn keine andere Transformationsregel mehr anwendbar ist. Hierbei sollte man jedes kritische Paar natürlich nur einmal im Verlauf der Transformation erzeugen. Um Fehlschlag zu erkennen, kann man z.B. die Heuristik verwenden, dass man dann abbrechen sollte, wenn alle kritischen Paare mit einer bestimmten Regel gebildet wurden und soweit wie möglich in Regeln überführt und reduziert wurden, und dennoch noch nicht-orientierbare Gleichungen übrig bleiben. Aufgrund der verlangten Fairness der Transformationsfolge muss man natürlich sicher stellen, dass alle kritischen Paare der persistenten Regeln irgendwann einmal als Gleichungen auftraten. (Es gibt darüber hinaus auch verbesserte Kriterien, die es erlauben, auf eine erneute kritische Paar-Bildung zwischen Regeln zu verzichten, die sich nur aufgrund der "Reduziere-Rechts"-Transformationsregel verändert habe, vgl. [Hue80] und [BN98, Abschnitt 7.4].)

Beispiel 6.2.13 Wir zeigen nun, wie das Gruppenbeispiel vervollständigt wird. Wir beginnen wie in Bsp. 6.2.1. Aus den ursprünglichen Gleichungen

$$\mathcal{E} = \{\mathsf{f}(x, \mathsf{f}(y, z)) \equiv \mathsf{f}(\mathsf{f}(x, y), z), \mathsf{f}(x, \mathsf{e}) \equiv x, \mathsf{f}(x, \mathsf{i}(x)) \equiv \mathsf{e}\}$$

entsteht durch "Orientieren" das folgende TES. Hierbei wird die RPOS mit Präzedenz $\mathsf{f} \sqsupset \mathsf{e}$ verwendet, wobei das Funktionssymbol f den Status $\langle 2, 1\rangle$ hat.

$$\begin{array}{rcll} \mathsf{f}(x, \mathsf{f}(y, z)) & \to & \mathsf{f}(\mathsf{f}(x, y), z) & (G1) \\ \mathsf{f}(x, \mathsf{e}) & \to & x & (G2) \\ \mathsf{f}(x, \mathsf{i}(x)) & \to & \mathsf{e} & (G3) \end{array}$$

Durch das kritische Paar zwischen $(G1)$ und $(G3)$ entsteht durch "Generieren", "Reduziere-Gleichung" und "Orientieren" die neue Regel

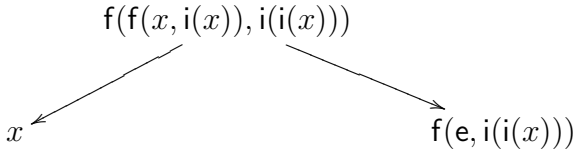
$$\mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) \;\to\; x \quad (G4)$$

Ebenso entsteht aus $(G4)$ und $(G2)$ die Regel $(G5)$, aus $(G4)$ und $(G5)$ die Regel $(G6)$ und aus $(G3)$ und $(G6)$ die Regel $(G7)$.

$$\begin{aligned} \mathsf{f}(x,\mathsf{i}(\mathsf{e})) &\to x \quad (G5)\\ \mathsf{f}(x,\mathsf{i}(\mathsf{i}(\mathsf{e}))) &\to x \quad (G6)\\ \mathsf{i}(\mathsf{e}) &\to \mathsf{e} \quad (G7) \end{aligned}$$

Durch "Reduziere-Links" wird nun $(G5)$ mit Hilfe von $(G7)$ in die Gleichung $\mathsf{f}(x,\mathsf{e}) \equiv x$ überführt, die dann durch "Reduziere-Gleichung" zu einer trivialen Identität $x \equiv x$ reduziert wird und mit "Löschen" gelöscht wird. Auf analoge Weise wird auch $(G6)$ gelöscht.

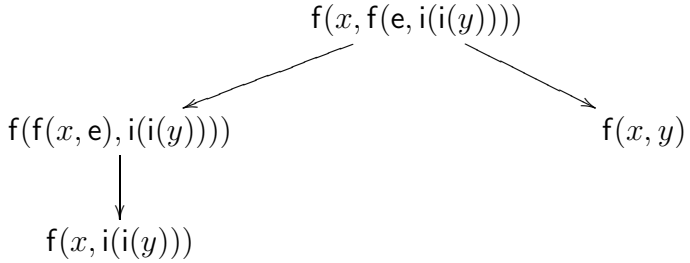
Die Überlagerung von $(G4)$ und $(G3)$ ergibt die kritische Situation



und die neue Regel:

$$\mathsf{f}(\mathsf{e},\mathsf{i}(\mathsf{i}(x))) \;\to\; x \quad (G8)$$

Die Überlagerung von $(G1)$ und $(G8)$ führt zu



und der neuen Regel:

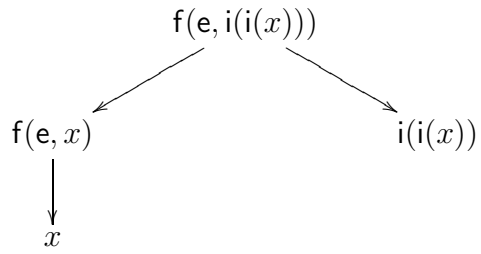
$$\mathsf{f}(x,\mathsf{i}(\mathsf{i}(y))) \;\to\; \mathsf{f}(x,y) \quad (G9)$$

Durch Anwendung von "Reduziere-Links" wird $(G8)$ überführt in $\mathsf{f}(\mathsf{e},x) \equiv x$ und "Orientieren" ergibt die Regel

$$\mathsf{f}(\mathsf{e},x) \;\to\; x \quad (G10)$$

Wir haben nun also herausgefunden, dass e auch das linksneutrale Element zu f ist.

Wir bilden nun das kritische Paar von $(G9)$ und $(G10)$.

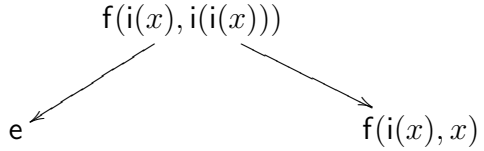


Dies ergibt

$$\mathsf{i}(\mathsf{i}(x)) \quad \to \quad x \quad (G11)$$

Nun haben wir also entdeckt, dass die inverse Funktion i in der Tat selbstinvers ist. Durch die Regel $(G11)$ lässt sich $(G9)$ nun reduzieren und anschließend löschen.

Nun überlagern wir $(G3)$ und $(G11)$.

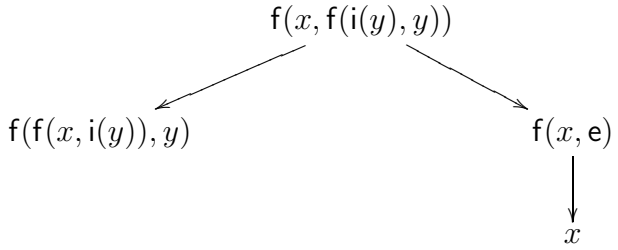


So erhält man die Regel

$$\mathsf{f}(\mathsf{i}(x), x) \quad \to \quad \mathsf{e} \quad (G12)$$

die besagt, dass i auch linksinvers ist.

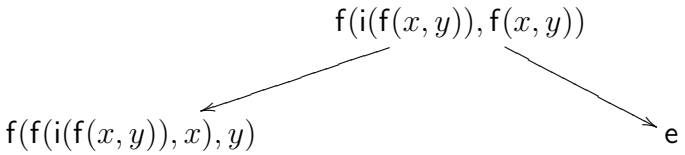
Die Überlagerung von $(G1)$ und $(G12)$ (an der Stelle 2 in $(G1)$s linker Seite) ergibt



und die Regel

$$\mathsf{f}(\mathsf{f}(x, \mathsf{i}(y)), y) \quad \to \quad x \quad (G13)$$

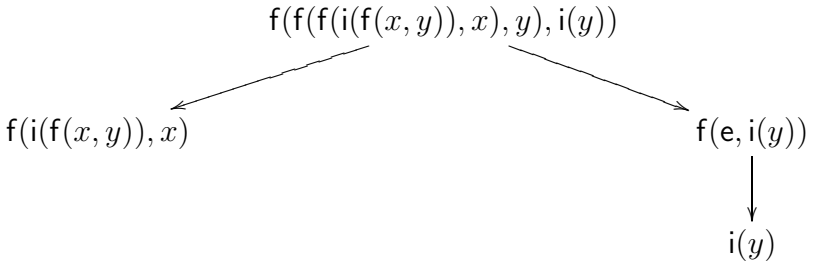
Die Überlagerung von $(G1)$ und $(G12)$ an den Positionen ϵ führt zu



und der Regel

$$\mathsf{f}(\mathsf{f}(\mathsf{i}(\mathsf{f}(x, y)), x), y) \quad \to \quad \mathsf{e} \quad (G14)$$

Nun überlagern wir $(G4)$ und $(G14)$ (an der Stelle 1 in $(G4)$s linker Seite).

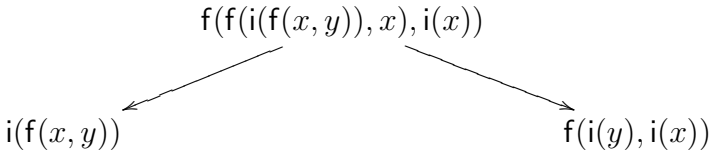


Dies führt zur Regel

$$\mathsf{f}(\mathsf{i}(\mathsf{f}(x,y)),x) \;\to\; \mathsf{i}(y) \quad (G15)$$

Durch "Reduziere-Links" kann man nun $(G14)$ zur Gleichung $\mathsf{f}(\mathsf{i}(y),y) \equiv \mathsf{e}$ reduzieren, die anschließend durch weitere Reduktion gelöscht werden kann.

Schließlich überlagern wir $(G4)$ und $(G15)$, was zur kritischen Situation



und (bei einer Präzedenz $\mathsf{i} \sqsupset \mathsf{f}$) zur Regel

$$\mathsf{i}(\mathsf{f}(x,y)) \;\to\; \mathsf{f}(\mathsf{i}(y),\mathsf{i}(x)) \quad (G16)$$

führt. Nun kann man $(G15)$ zu $\mathsf{f}(\mathsf{f}(\mathsf{i}(y),\mathsf{i}(x)),x) \equiv \mathsf{i}(y)$ reduzieren und mit $(G13)$ kann man diese Gleichung dann zu einer trivialen Gleichung vereinfachen, die anschließend gelöscht wird.

Es stellt sich heraus, dass nun alle verbleibenden kritischen Paare zusammenführbar sind. Somit haben wir nun ein konvergentes TES erzeugt, dass zu den ursprünglichen Gruppenaxiomen äquivalent ist. Mit diesem TES hat man ein Entscheidungsverfahren für alle Gleichheitsaussagen über Gruppen zur Verfügung. Wie im Beispiel deutlich wurde, kann man solch eine Vervollständigung leicht automatisch durchführen.

$$\begin{array}{rcll}
\mathsf{f}(x,\mathsf{f}(y,z)) & \to & \mathsf{f}(\mathsf{f}(x,y),z) & (G1)\\
\mathsf{f}(x,\mathsf{e}) & \to & x & (G2)\\
\mathsf{f}(x,\mathsf{i}(x)) & \to & \mathsf{e} & (G3)\\
\mathsf{f}(\mathsf{f}(x,y),\mathsf{i}(y)) & \to & x & (G4)\\
\mathsf{i}(\mathsf{e}) & \to & \mathsf{e} & (G7)
\end{array}
\qquad
\begin{array}{rcll}
\mathsf{f}(\mathsf{e},x) & \to & x & (G10)\\
\mathsf{i}(\mathsf{i}(x)) & \to & x & (G11)\\
\mathsf{f}(\mathsf{i}(x),x) & \to & \mathsf{e} & (G12)\\
\mathsf{f}(\mathsf{f}(x,\mathsf{i}(y)),y) & \to & x & (G13)\\
\mathsf{i}(\mathsf{f}(x,y)) & \to & \mathsf{f}(\mathsf{i}(y),\mathsf{i}(x)) & (G16)
\end{array}$$

6.3 Implizite Induktion

Wir haben uns bisher mit der Lösung des Wortproblems beschäftigt, d.h., mit der Frage, ob eine Gleichung $s \equiv t$ in *allen* Modellen eines Gleichungssystems $\mathcal{E}$ gilt. In der Programmverifikation ist man jedoch meist nicht an der Gültigkeit einer Aussage in allen Modellen einer Menge von Axiomen $\mathcal{E}$ interessiert, sondern man möchte die Gültigkeit von Gleichungen in einem speziellen Modell von $\mathcal{E}$ beweisen.

Beispiel 6.3.1 Beispielsweise bildet die Algebra $(\mathbb{N},\alpha)$ mit $\alpha_{\mathcal{O}} = 0$, $\alpha_{\mathsf{succ}}(n) = n+1$ und $\alpha_{\mathsf{plus}}(n,m) = n+m$ ein Modell der folgenden Gleichungsmenge $\mathcal{E}$:

$$\begin{array}{rcll}
\mathsf{plus}(\mathcal{O},y) & \equiv & y & (6.1)\\
\mathsf{plus}(\mathsf{succ}(x),y) & \equiv & \mathsf{succ}(\mathsf{plus}(x,y)) & (6.2)
\end{array}$$

Die Aussage $\mathsf{plus}(x,\mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x,y))$ ist in der obigen Algebra gültig, da sie ja eine wahre Aussage über die Adddition natürlicher Zahlen ist. Sie folgt aber nicht aus

den Gleichungen $\mathcal{E}$, d.h., $\mathcal{E} \not\models \mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$. Der Grund ist, dass es Modelle von $\mathcal{E}$ gibt, in denen diese Gleichung nicht gilt. Ein Beispiel ist die Algebra $B = (\mathbb{N} \cup \{\Box, \diamond\}, \alpha')$ mit $\alpha'_{\mathcal{O}} = 0$ und

$$\alpha'_{\mathsf{succ}}(n) = \begin{cases} n+1, & \text{falls } n \in \mathbb{N} \\ \diamond, & \text{falls } n \in \{\Box, \diamond\} \end{cases}$$

$$\alpha'_{\mathsf{plus}}(n, m) = \begin{cases} n+m, & \text{falls } n, m \in \mathbb{N} \\ n, & \text{falls } n \in \{\Box, \diamond\} \\ m, & \text{falls } n = 0 \text{ und } m \in \{\Box, \diamond\} \\ \diamond, & \text{falls } n > 0 \text{ und } m \in \{\Box, \diamond\} \end{cases}$$

Es gilt $B \not\models \mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$. Um dies zu sehen, sei I die Interpretation, die aus B und der Variablenbelegung β mit $\beta(x) = \Box$ und $\beta(y) = 0$ entsteht. Dann gilt $I(\mathsf{plus}(x, \mathsf{succ}(y))) = \alpha'_{\mathsf{plus}}(\Box, \alpha'_{\mathsf{succ}}(0)) = \Box$ und $I(\mathsf{succ}(\mathsf{plus}(x, y))) = \alpha'_{\mathsf{succ}}(\alpha'_{\mathsf{plus}}(\Box, 0)) = \alpha'_{\mathsf{succ}}(\Box) = \diamond$.

Würde man die Gleichungen (6.1) und (6.2) von links nach rechts orientieren und das entstehende TES als funktionales Programm ansehen, so hätte man damit ein Programm zur Berechnung der Addition natürlicher Zahlen. Die Untersuchung, ob dieses Programm eine bestimmte (Teil-)Spezifikation wie $\mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$ erfüllt, ist die Frage der *Programmverifikation*. Natürlich liefern die Terme $\mathsf{plus}(x, \mathsf{succ}(y))$ und $\mathsf{succ}(\mathsf{plus}(x, y))$ *bei jeder Instantiierung der Variablen x und y mit natürlichen Zahlen* (d.h., mit Termen wie $\mathcal{O}$, $\mathsf{succ}(\mathcal{O})$, $\mathsf{succ}^2(\mathcal{O})$, etc.) dasselbe Ergebnis. Mit anderen Worten, $\mathsf{plus}(x, \mathsf{succ}(y))\sigma$ und $\mathsf{succ}(\mathsf{plus}(x, y))\sigma$ reduzieren zu derselben Normalform, sofern σ die Variablen mit Termen wie oben belegt. Daher würde man diese Aussage über das plus-Programm als "korrekt" bezeichnen.

Man erkennt also, dass bei der Programmverifikation typischerweise nicht die Gültigkeit einer Gleichung in allen Modellen des Programms (bzw. seiner definierenden Gleichungen) gemeint ist. Stattdessen betrachtet man nur solche Modelle, bei denen alle Elemente des Trägers auch durch Grundterme repräsentiert werden können. Falls eine Aussage in allen solchen Modellen gilt, dann bezeichnet man sie als *induktiv gültig*.

Definition 6.3.2 (Induktive Gültigkeit) *Sei $\mathcal{E}$ ein Gleichungssystem über Σ und $\mathcal{V}$, seien $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$. Die Gleichung $s \equiv t$ ist* induktiv gültig *in $\mathcal{E}$ ($\mathcal{E} \models_I s \equiv t$) gdw. $\mathcal{E} \models s\sigma \equiv t\sigma$ für alle Substitutionen σ gilt, bei denen $\sigma(x) \in \mathcal{T}(\Sigma)$ für alle $x \in \mathcal{V}(s) \cup \mathcal{V}(t)$ gilt (d.h., für alle Substitutionen, die die Variablen von s und t mit Grundtermen belegen).*

Offensichtlich ist jede aus $\mathcal{E}$ folgerbare (d.h. gültige) Gleichung auch induktiv gültig (d.h., aus $\mathcal{E} \models s \equiv t$ folgt $\mathcal{E} \models_I s \equiv t$), aber die Rückrichtung gilt nicht. In Bsp. 6.3.1 wurde gezeigt, dass die Gleichung $\mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$ nicht gültig in $\mathcal{E} = \{(6.1), (6.2)\}$ ist. Wir werden nun nachweisen, dass sie aber induktiv gültig ist. Hierzu muss man zeigen, dass die Grundgleichung $\mathsf{plus}(t_1, \mathsf{succ}(t_2)) \equiv \mathsf{succ}(\mathsf{plus}(t_1, t_2))$ für alle Grundterme t_1 und t_2 aus $\mathcal{E}$ folgt.

Beispiel 6.3.3 Dieser Beweis ließe sich durch strukturelle Induktion über t_1 führen. Hierbei gehen wir von $\Sigma = \{\mathcal{O}, \mathsf{succ}, \mathsf{plus}\}$ aus und verwenden das TES $\mathcal{R}$, das durch Richten der Gleichungen von links nach rechts entsteht.

Man erkennt, dass jeder Grundterm aus $\mathcal{T}(\Sigma)$ zu einem Grundterm aus $\mathcal{T}(\{\mathcal{O}, \mathsf{succ}\})$ reduziert werden kann. Der Grund ist, dass die linken Seiten $\mathsf{plus}(\mathcal{O}, y)$ und $\mathsf{plus}(\mathsf{succ}(x), y)$ der beiden plus-Regeln alle möglichen Eingaben abdecken (d.h., plus ist vollständig definiert) und dass $\mathcal{R}$ terminiert. Daher betrachten wir im folgenden nur noch Grundterme t_1 aus $\mathcal{T}(\{\mathcal{O}, \mathsf{succ}\})$.

Falls $t_1 = \mathcal{O}$ ist, so reduzieren $\mathsf{plus}(\mathcal{O}, \mathsf{succ}(t_2))$ und $\mathsf{succ}(\mathsf{plus}(\mathcal{O}, t_2))$ beide zu $\mathsf{succ}(t_2)$. Falls $t_1 = \mathsf{succ}(t_1')$ ist, so gilt $\mathsf{plus}(\mathsf{succ}(t_1'), \mathsf{succ}(t_2)) \to_{\mathcal{R}} \mathsf{succ}(\underline{\mathsf{plus}(t_1', \mathsf{succ}(t_2))})$ und $\mathsf{succ}(\mathsf{plus}(\mathsf{succ}(t_1'), t_2)) \to_{\mathcal{R}} \mathsf{succ}(\underline{\mathsf{succ}(\mathsf{plus}(t_1', t_2))})$. Da nach der Induktionshypothese $\mathsf{plus}(t_1', \mathsf{succ}(t_2)) \equiv_{\mathcal{E}} \mathsf{succ}(\mathsf{plus}(t_1', t_2))$ gilt, müssen diese Terme zu einem Term q zusammenführbar sein. Damit sind dann auch die ursprünglichen Terme zum Term $\mathsf{succ}(q)$ zusammenführbar.

Der obige Induktionsbeweis entspricht einem Beweis mit der Peano-Induktion über natürliche Zahlen. In allen bislang betrachteten Induktionsbeweisen wurde explizit eine Induktionsrelation gebildet und Induktionshypothesen wurden explizit angewendet. Der Nachteil bei einer Automatisierung dieses Verfahrens besteht darin, dass dann automatisch entschieden werden muss, welche Induktionsrelation bzw. welche Induktionsvariablen gewählt werden sollen und wann Induktionshypothesen auf welche Art verwendet werden sollen. Genauere Details und Techniken zu diesem Ansatz finden sich z.B. in [Gie03].

Aus diesem Grund stellen wir in diesem Abschnitt eine alternative Möglichkeit vor, um Induktionsbeweise zu automatisieren. Die Idee besteht darin, einfach den Vervollständigungsalgorithmus der vorangegangenen Abschnitte zu verwenden und keine Induktionsrelationen, Induktionsvariablen, Induktionshypothesen etc. zu bilden. Daher bezeichnet man diese Form der Induktion als *implizit*.

Die Kernidee zum Nachweis von induktiver Gültigkeit auf diese Weise ist die sogenannte *Konsistenzbeweismethode*: Eine Gleichung $s \equiv t$ ist genau dann induktiv gültig in $\mathcal{E}$, wenn aus $\mathcal{E} \cup \{s \equiv t\}$ die gleichen Grundgleichungen wie schon aus $\mathcal{E}$ folgen. Mit anderen Worten, die Hinzunahme der Gleichung $s \equiv t$ zu $\mathcal{E}$ darf keine "Inkonsistenzen" erzeugen.

Satz 6.3.4 (Konsistenzbeweismethode) *Sei $\mathcal{E}$ ein Gleichungssystem über Σ und $\mathcal{V}$, seien $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$. Dann gilt $\mathcal{E} \models_I s \equiv t$ gdw. für alle Grundterme $u, v \in \mathcal{T}(\Sigma)$ mit $\mathcal{E} \not\models u \equiv v$ auch $\mathcal{E} \cup \{s \equiv t\} \not\models u \equiv v$ gilt.*

Beweis. Wir zeigen zunächst die Richtung von links nach rechts. Hierbei setzen wir voraus, dass $s \equiv t$ induktiv gültig ist. Zu zeigen ist nun, dass aus $\mathcal{E} \cup \{s \equiv t\} \models u \equiv v$ auch $\mathcal{E} \models u \equiv v$ folgt. Nach dem Satz von Birkhoff (Satz 3.1.14) bedeutet $\mathcal{E} \cup \{s \equiv t\} \models u \equiv v$, dass es eine Herleitung $u = u_0 \leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}} u_1 \leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}} \ldots \leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}} u_n = v$ gibt. Sei δ eine Substitution, die alle Variablen in den u_i durch Grundterme ersetzt. Dann gilt aufgrund der Stabilität von $\leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}}$ (Lemma 3.1.13) auch $u = u\delta = u_0\delta \leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}} u_1\delta \leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}} \ldots \leftrightarrow_{\mathcal{E} \cup \{s \equiv t\}} u_n\delta = v\delta = v$. In dieser Herleitung werden nur Grundinstanzen von s durch Grundinstanzen von t (und umgekehrt) ersetzt. Daher lässt sich die obige Herleitung also auch mit der Ersetzungsrelation der unendlichen Gleichungsmenge $\mathcal{E} \cup \{s\sigma \equiv t\sigma \mid x\sigma \in \mathcal{T}(\Sigma) \text{ für alle } x \in \mathcal{V}(s) \cup \mathcal{V}(t)\}$ durchführen. Nach dem Satz von Birkhoff (der auch für

unendliche Gleichungsmengen gilt), hat man damit auch $\mathcal{E} \cup \{s\sigma \equiv t\sigma \mid x\sigma \in \mathcal{T}(\Sigma)$ für alle $x \in \mathcal{V}(s) \cup \mathcal{V}(t)\} \models u \equiv v$. Da aufgrund der Voraussetzung $\mathcal{E} \models_I s \equiv t$ jedes Modell von $\mathcal{E}$ auch Modell von $\{s\sigma \equiv t\sigma \mid x\sigma \in \mathcal{T}(\Sigma)$ für alle $x \in \mathcal{V}(s) \cup \mathcal{V}(t)\}$ ist, folgt damit auch $\mathcal{E} \models u \equiv v$.

Für die Richtung von rechts nach links setzen wir voraus, dass aus $\mathcal{E} \cup \{s \equiv t\} \models u \equiv v$ jeweils auch $\mathcal{E} \models u \equiv v$ folgt. Man erkennt, dass $\mathcal{E} \cup \{s \equiv t\} \models s\sigma \equiv t\sigma$ für alle Substitutionen σ gilt. Damit gilt dies insbesondere für die Substitutionen, bei denen $s\sigma$ und $t\sigma$ Grundterme sind. Nach Voraussetzung gilt für diese Substitutionen dann auch $\mathcal{E} \models s\sigma \equiv t\sigma$. Dies bedeutet, dass $s \equiv t$ in der Tat induktiv gültig in $\mathcal{E}$ ist. □

Für die obige Konsistenzbeweismethode muss man also überprüfen, ob durch die Hinzunahme der Gleichung $s \equiv t$ neue Gleichungen $u \equiv v$ gelten, die bislang nicht aus $\mathcal{E}$ folgten. Um dies zu automatisieren, versuchen wir, ein zu $\mathcal{E}$ äquivalentes konvergentes Termersetzungssystem $\mathcal{R}$ zu finden. Hierzu verwenden wir die Vervollständigungsverfahren aus den vorigen Abschnitten. Der folgende Satz zeigt, dass man dann nur untersuchen muss, ob es verschiedene Grundnormalformen $q_1 \neq q_2$ von $\mathcal{R}$ gibt, deren Gleichheit aus $\mathcal{E} \cup \{s \equiv t\}$ folgt. Hierbei bezeichnen wir die Menge der Grundnormalformen eines TES $\mathcal{R}$ als $\mathsf{NF}(\mathcal{R}) = \{q\downarrow_{\mathcal{R}} \mid q \in \mathcal{T}(\Sigma)\}$. Wenn $\mathcal{R}$ beispielsweise das TES ist, das durch Richten der Gleichungen (6.1) und (6.2) von links nach rechts entsteht, dann ergibt sich $\mathsf{NF}(\mathcal{R}) = \{\mathcal{O}, \mathsf{succ}(\mathcal{O}), \mathsf{succ}^2(\mathcal{O}), \ldots\}$.

Satz 6.3.5 (Konsistenzbeweismethode mit konvergenten TESen) *Sei $\mathcal{E}$ ein Gleichungssystem über Σ und $\mathcal{V}$, sei $\mathcal{R}$ ein zu $\mathcal{E}$ äquivalentes konvergentes TES, seien $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$. Dann gilt $\mathcal{E} \models_I s \equiv t$ gdw. für alle $q_1, q_2 \in \mathsf{NF}(\mathcal{R})$ mit $q_1 \neq q_2$ auch $\mathcal{E} \cup \{s \equiv t\} \not\models q_1 \equiv q_2$ gilt.*

Beweis. Wir zeigen zuerst die Richtung von links nach rechts. Wir nehmen an, dass $\mathcal{E} \cup \{s \equiv t\} \models q_1 \equiv q_2$ für verschiedene Grundnormalformen q_1 und q_2 von $\mathcal{R}$ gilt. Da $s \equiv t$ nach Voraussetzung induktiv gültig ist, erhält man damit $\mathcal{E} \models q_1 \equiv q_2$ nach Satz 6.3.4. Die Konvergenz und Äquivalenz von $\mathcal{R}$ und $\mathcal{E}$ ergibt dann $q_1 \downarrow_{\mathcal{R}} q_2$ und somit $q_1 = q_2$, da es sich um Normalformen handelt. Dies ist ein Widerspruch zur Voraussetzung.

Für die Rückrichtung nehmen wir $\mathcal{E} \not\models_I s \equiv t$ an. Nach Satz 6.3.4 existieren also Grundterme u, v, so dass $\mathcal{E} \cup \{s \equiv t\} \models u \equiv v$ und $\mathcal{E} \not\models u \equiv v$. Sei $q_1 = u\downarrow_{\mathcal{R}}$ und $q_2 = v\downarrow_{\mathcal{R}}$. Diese Normalformen existieren aufgrund der Konvergenz von $\mathcal{R}$. Wegen der Äquivalenz von $\mathcal{R}$ und $\mathcal{E}$ gilt $q_1 \neq q_2$. Da $\mathcal{R}$ korrekt für $\mathcal{E}$ und somit auch für $\mathcal{E} \cup \{s \equiv t\}$ ist, gilt außerdem $\mathcal{E} \cup \{s \equiv t\} \models q_1 \equiv u \equiv v \equiv q_2$. Dies widerspricht der Voraussetzung. □

Falls die Konstruktion des konvergenten und zu $\mathcal{E}$ äquivalenten TES $\mathcal{R}$ also mit dem Vervollständigungsalgorithmus gelingt, so muss man nun für die Konsistenzbeweismethode überprüfen, ob verschiedene Grundnormalformen von $\mathcal{R}$ bezüglich $\mathcal{E} \cup \{s \equiv t\}$ gleich sind. Hierzu stellt sich das Problem, dass $\mathsf{NF}(\mathcal{R})$ im allgemeinen unendlich ist. Bei dem plus-TES aus Bsp. 6.3.1 haben wir beispielsweise $\mathsf{NF}(\mathcal{R}) = \{\mathcal{O}, \mathsf{succ}(\mathcal{O}), \mathsf{succ}^2(\mathcal{O}), \ldots\}$.

Die Lösung besteht darin, sich auf TESe $\mathcal{R}$ bestimmter Gestalt zu beschränken, bei denen man die Menge $\mathsf{NF}(\mathcal{R})$ auf endliche Weise repräsentieren kann. Der Testschritt, ob jeweils $\mathcal{E} \cup \{s \equiv t\} \models q_1 \equiv q_2$ gilt, wird dann in den Vervollständigungsalgorithmus integriert.

Die Grundidee für die Erkennung und Repräsentation der Grundnormalformen besteht darin, bestimmte Funktionssymbole (sogenannte *Konstruktoren*) auszuzeichnen, mit denen die in $\mathcal{E}$ verschiedenen Grundterme gebildet werden können. Die Konstruktoren dienen also zum Aufbau aller "Datenobjekte", über die die Gleichungen in $\mathcal{E}$ etwas aussagen. Wir teilen daher die Signatur Σ in zwei disjunkte Teilmengen Σ^c und Σ^d für die Konstruktoren und die *definierten Funktionssymbole* auf, die den durch Gleichungen festgelegten Symbolen bzw. den Algorithmen entsprechen. Die Bedingungen, die wir nun an das zu $\mathcal{E}$ äquivalente TES $\mathcal{R}$ stellen, besagen, dass $\mathcal{R}$ keine Regeln nur für Konstruktoren enthalten darf und dass definierte Funktionssymbole vollständig definiert sein müssen. Dann sagen wir, dass $\mathcal{R}$ das "Definitionsprinzip" erfüllt.

Definition 6.3.6 (Definitionsprinzip) *Sei $\Sigma^c \subseteq \Sigma$ eine Teilmenge der Signatur (die Menge der* Konstruktoren*) mit $\Sigma^c_0 \neq \varnothing$ (d.h., es gibt mindestens einen konstanten Konstruktor) und $|\Sigma^c| \geq 2$ (d.h., es gibt mindestens zwei verschiedene Konstruktoren). Sei $\Sigma^d = \Sigma \setminus \Sigma^c$ die Menge der* definierten Funktionssymbole*. Ein TES $\mathcal{R}$ über Σ und $\mathcal{V}$ genügt dem* Definitionsprinzip *für die Menge der Konstruktoren Σ^c gdw. die beiden folgenden Bedingungen erfüllt sind:*

(a) Für alle Regeln $l \to r \in \mathcal{R}$ gilt $l \notin \mathcal{T}(\Sigma^c, \mathcal{V})$, d.h., l enthält mindestens ein definiertes Funktionssymbol.

(b) Für jedes $f \in \Sigma^d$ und alle $t_1, \ldots, t_n \in \mathcal{T}(\Sigma^c)$ existiert eine Regel $l \to r \in \mathcal{R}$, so dass l den Term $f(t_1, \ldots, t_n)$ matcht.

Die Bedingung (a) ist leicht überprüfbar und für Bedingung (b) existieren offensichtliche, leicht zu überprüfende hinreichende Bedingungen. Die Überprüfung von (b) ist besonders leicht, wenn $\mathcal{R}$ ein sogenanntes *linkslineares Konstruktorsystem* ist, d.h., ein TES, dessen linken Seiten nur an der äußersten Position ein definiertes Funktionssymbol haben, dessen Argumente dann nur noch Konstruktoren und Variablen enthalten, wobei keine Variable mehrfach in einer linken Seite auftritt. Diese Form von Regeln wird in den meisten funktionalen Programmiersprachen (z.B. in HASKELL) benutzt.

Beispiel 6.3.7 Das TES $\mathcal{R}$, das durch Richten der Gleichungen (6.1) und (6.2) von links nach rechts entsteht, erfüllt das Definitionsprinzip für die Menge der Konstruktoren $\Sigma^c = \{\mathcal{O}, \mathsf{succ}\}$. Jede linke Seite enthält das definierte Funktionssymbol plus und jeder Term der Gestalt $\mathsf{plus}(\mathsf{succ}^n(\mathcal{O}), \mathsf{succ}^m(\mathcal{O}))$ ist reduzierbar. Der Grund ist, dass die Patterns $(\mathcal{O}, y)$ und $(\mathsf{succ}(x), y)$ der beiden Regeln offensichtlich jeden dieser Fälle abdecken, d.h., plus ist vollständig definiert.

Der folgende Satz zeigt, dass die Grundnormalformen eines TES, das das Definitionsprinzip erfüllt, gerade die Konstruktorgrundterme sind.

Satz 6.3.8 (Grundnormalformen bei Definitionsprinzip) *Sei $\mathcal{R}$ ein TES, das das Definitionsprinzip für die Menge der Konstruktoren Σ^c erfüllt. Dann gilt $\mathsf{NF}(\mathcal{R}) = \mathcal{T}(\Sigma^c)$.*

Beweis. Um $\mathsf{NF}(\mathcal{R}) \subseteq \mathcal{T}(\Sigma^c)$ zu zeigen, nehmen wir an, dass es eine Grundnormalform q gibt, die ein definiertes Funktionssymbol enthält. Dann existieren auch "innerste" Vorkommen von definierten Funktionssymbolen, d.h., es existiert ein $\pi \in Occ(q)$, so dass $q|_\pi = f(t_1, \ldots, t_n)$ für $f \in \Sigma^d$ und $t_1, \ldots, t_n \in \mathcal{T}(\Sigma^c)$. Nach Def. 6.3.6 (b) ist dann $q|_\pi$ und damit auch q aber keine Normalform.

Ebenso gilt auch $\mathsf{NF}(\mathcal{R}) \supseteq \mathcal{T}(\Sigma^c)$. Der Grund ist, dass alle Konstruktorgrundterme Normalformen sind, da jede linke Seite einer Regel aus $\mathcal{R}$ ein definiertes Funktionssymbol enthält (wegen Def. 6.3.6 (a)). □

Zur Automatisierung der Konsistenzbeweismethode verwenden wir den Vervollständigungsalgorithmus, um ein zu $\mathcal{E} \cup \{s \equiv t\}$ äquivalentes und konvergentes TES zu konstruieren. Hierbei überprüfen wir, ob während der Vervollständigung neue Gleichungen entstehen, die verschiedene Konstruktorgrundterme gleich machen würden. In diesem Fall hätten wir gezeigt, dass die Gleichung $s \equiv t$ nicht induktiv gültig ist. Falls es uns hingegen gelingt, ein zu $\mathcal{E} \cup \{s \equiv t\}$ äquivalentes und konvergentes TES zu konstruieren, das keine verschiedenen Konstruktorgrundterme gleich macht, so ist die induktive Gültigkeit von $s \equiv t$ bewiesen. Wir gehen hierbei davon aus, dass das zu $\mathcal{E}$ äquivalente konvergente TES $\mathcal{R}$ dem Definitionsprinzip genügt. Bei der Konstruktion des TES, das zu $\mathcal{E} \cup \{s \equiv t\}$ äquivalent und konvergent sein soll, starten wir den verbesserten Vervollständigungsalgorithmus nun nicht mit $(\mathcal{E} \cup \{s \equiv t\}, \varnothing)$, sondern mit $(\{s \equiv t\}, \mathcal{R})$. Dies ist nach wie vor korrekt, da $\leftrightarrow^*_{\mathcal{E} \cup \{s \equiv t\}} = \leftrightarrow^*_{\{s \equiv t\} \cup \mathcal{R}}$ gilt.

Um das Vervollständigungsverfahren erfolgreich für Induktionsbeweise verwenden zu können, wird es geringfügig modifiziert und erweitert. Bei diesem modifizierten Verfahren kann sich dann während der Vervollständigung die Menge der gültigen Gleichungen verändern (und zwar vergrößern), aber sofern $\mathcal{E} \cup \{s \equiv t\}$ konsistent ist (d.h., aus $\mathcal{E} \cup \{s \equiv t\}$ folgen keine Gleichheiten zwischen verschiedenen Konstruktorgrundtermen), so bleibt die Menge der gültigen *Grundgleichungen* (d.h., Gleichungen zwischen Grundtermen) gleich. Wenn man also im Vervollständigungsverfahren von $(\mathcal{E}, \mathcal{R})$ zu $(\mathcal{E}', \mathcal{R}')$ kommt und $\mathcal{E} \cup \mathcal{R}$ konsistent ist, so sind die Relationen $\leftrightarrow^*_{\mathcal{E} \cup \mathcal{R}}$ und $\leftrightarrow^*_{\mathcal{E}' \cup \mathcal{R}'}$ auf *Grundtermen* gleich. Die Gleichheit von $\leftrightarrow^*_{\mathcal{E} \cup \mathcal{R}}$ und $\leftrightarrow^*_{\mathcal{E}' \cup \mathcal{R}'}$ auf Grundtermen ist für Induktionsbeweise ausreichend: Wie Satz 6.3.5 zeigt, genügt es, ein konvergentes TES zu konstruieren, das zu $\mathcal{E} \cup \{s \equiv t\}$ auf Grundtermen äquivalent ist und zu zeigen, dass dieses TES keine verschiedenen Konstruktorgrundterme zusammenführt.

In dem modifizierten Vervollständigungsverfahren wird sichergestellt, dass das durch die Vervollständigung entstehende TES konsistent bleibt. Hierzu wird die Orientierung von Gleichungen in Regeln nur noch dann erlaubt, wenn die neuen Regeln außen ein definiertes Funktionssymbol haben.

Um Inkonsistenzen zu entdecken, wird der Vervollständigungsalgorithmus außerdem um zwei Regeln ergänzt. Sie behandeln den Fall, dass während der Vervollständigung eine Gleichung entsteht, bei der kein äußeres Funktionssymbol definiert ist. Hierbei gibt es die folgenden Möglichkeiten: Falls eine Gleichung der Form $c_1(\ldots) \equiv c_2(\ldots)$, $c_1(\ldots) \equiv x$ oder $x \equiv c_2(\ldots)$ für Konstruktoren $c_1 \neq c_2$ entsteht, so folgt daraus, dass verschiedene Konstruktorgrundterme gleich sein müssen und wir haben daher eine Inkonsistenz entdeckt. Die Aussage $s \equiv t$ ist also nicht induktiv gültig und wir brechen mit "*False*" ab.

Falls wir eine Gleichung der Form $c(s_1, \ldots, s_n) \equiv c(t_1, \ldots, t_n)$ erhalten, so verwenden wir die *Injektivität* der Konstruktoren und ersetzen die ursprüngliche Gleichung durch die Gleichungen $s_1 \equiv t_1, \ldots, s_n \equiv t_n$. Man kann zeigen, dass $\leftrightarrow^*_{\mathcal{E} \cup \{c(s_1,\ldots,s_n) \equiv c(t_1,\ldots,t_n)\}}$ und $\leftrightarrow^*_{\mathcal{E} \cup \{s_1 \equiv t_1, \ldots, s_n \equiv t_n\}}$ auf Grundtermen gleich sind, sofern $\mathcal{E} \cup \{c(s_1, \ldots, s_n) \equiv c(t_1, \ldots, t_n)\}$ nicht inkonsistent ist.

Definition 6.3.9 (Induktionsbeweisverfahren [HH82]) *Sei $\mathcal{E}$ ein Gleichungssystem und $\mathcal{R}$ ein TES über Σ und $\mathcal{V}$, sei $\Sigma^c \subseteq \Sigma$, und sei $\succ$ eine Reduktionsordnung. Dann definieren wir den* Induktionsbeweiskalkül *als die sechs Regeln des Vervollständigungsverfahrens von Def. 6.2.2, wobei die Regel "*Orientieren*" wie folgt geändert wird und die weiteren Regeln "*Inkonsistenz*" und "*Injektivität*" ergänzt werden:*

Orientieren

$$\frac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \to t\}} \quad \textit{falls } s \succ t \textit{ und } s = f(\ldots) \textit{ mit } f \notin \Sigma^c$$

$$\frac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \to s\}} \quad \textit{falls } t \succ s \textit{ und } t = f(\ldots) \textit{ mit } f \notin \Sigma^c$$

Inkonsistenz

$$\frac{\mathcal{E} \cup \{s \equiv t\}, \mathcal{R}}{\text{"False"}}$$

falls $s = c_1(\ldots)$, $t = c_2(\ldots)$ für $c_1, c_2 \in \Sigma^c$ mit $c_1 \neq c_2$
oder $s = c(\ldots)$ und $t \in \mathcal{V}$ für $c \in \Sigma^c$
oder $t = c(\ldots)$ und $s \in \mathcal{V}$ für $c \in \Sigma^c$

Injektivität

$$\frac{\mathcal{E} \cup \{c(s_1, \ldots, s_n) \equiv c(t_1, \ldots, t_n)\}, \mathcal{R}}{\mathcal{E} \cup \{s_1 \equiv t_1, \ldots, s_n \equiv t_n\}, \mathcal{R}} \quad \textit{falls } c \in \Sigma^c$$

Wir schreiben $(\mathcal{E}, \mathcal{R}) \vdash_I (\mathcal{E}', \mathcal{R}')$, falls $(\mathcal{E}, \mathcal{R})$ mit Hilfe einer dieser Transformationsregeln in $(\mathcal{E}', \mathcal{R}')$ überführt werden kann.

Wir illustrieren die Anwendung der Transformationsregeln des Induktionsbeweisverfahrens nun an unserem Beispiel.

Beispiel 6.3.10 In der folgenden Tabelle geben wir jeweils den Namen der angewendeten Transformationsregel sowie die jeweilige Menge von Gleichungen und Regeln an. Hierbei haben wir die beiden ursprünglichen plus-Regeln aus $\mathcal{R}$ nicht mehr mit aufgeführt, um die Tabelle klein zu halten. Wir verwenden zur Orientierung eine LPO oder RPO mit der Präzedenz $\mathsf{plus} \sqsupset \mathsf{succ}$. Die beiden "Generieren"-Schritte bilden kritische Paare zwischen der neuen Regel $\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$ und der ersten Regel $\mathsf{plus}(\mathcal{O}, y) \to y$ bzw. der zweiten Regel $\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathsf{plus}(x, y))$ von $\mathcal{R}$.

Regel	$\mathcal{E}_i$	$\mathcal{R}_i \setminus \mathcal{R}$
	$\mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$	
Orientieren		$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Generieren	$\mathsf{succ}(y) \equiv \mathsf{succ}(\mathsf{plus}(\mathcal{O}, y))$	$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Reduz.-Gleichung	$\mathsf{succ}(y) \equiv \mathsf{succ}(y)$	$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Löschen		$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Generieren	$\mathsf{succ}(\mathsf{plus}(x, \mathsf{succ}(y))) \equiv \mathsf{succ}(\mathsf{plus}(\mathsf{succ}(x), y))$	$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Reduz.-Gleichung	$\mathsf{succ}(\mathsf{succ}(\mathsf{plus}(x, y))) \equiv \mathsf{succ}(\mathsf{plus}(\mathsf{succ}(x), y))$	$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Reduz.-Gleichung	$\mathsf{succ}(\mathsf{succ}(\mathsf{plus}(x, y))) \equiv \mathsf{succ}(\mathsf{succ}(\mathsf{plus}(x, y)))$	$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$
Löschen		$\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))$

Die Ableitung terminiert also mit $(\varnothing, \mathcal{R} \cup \{\mathsf{plus}(x, \mathsf{succ}(y)) \to \mathsf{succ}(\mathsf{plus}(x, y))\})$, denn es sind alle kritischen Paare betrachtet worden. Dies beweist $\mathcal{E} \models_I \mathsf{plus}(x, \mathsf{succ}(y)) \equiv \mathsf{succ}(\mathsf{plus}(x, y))$.

Auf analoge Weise lässt sich nun z.B. auch zeigen, dass plus eine assoziative Funktion berechnet, vgl. die Aussage (1.4) aus der Einleitung. Dies ist ebenfalls eine Aussage, die nicht aus den beiden Gleichungen (6.1) und (6.2) über plus folgt, die aber induktiv gültig ist. Das nächste Beispiel illustriert das Verhalten des Induktionsbeweisverfahrens bei einer Gleichung, die nicht induktiv gültig ist.

Beispiel 6.3.11 Wir betrachten dieselbe Gleichungsmenge $\mathcal{E}$ und dasselbe TES $\mathcal{R}$, das die beiden Regeln für plus enthält. Nun wollen wir untersuchen, ob $\mathcal{E} \models_I \mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathcal{O})$ gilt. Es ergibt sich die folgende Ableitung bei einer LPO oder RPO wie oben, bei der außerdem noch plus größer als $\mathcal{O}$ in der Präzedenz ist. Die beiden "Generieren"-Schritte bilden kritische Paare zwischen der neuen Regel $\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathcal{O})$ und der zweiten Regel $\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathsf{plus}(x, y))$ bzw. der ersten Regel $\mathsf{plus}(\mathcal{O}, y) \to y$ von $\mathcal{R}$.

Regel	$\mathcal{E}_i$	$\mathcal{R}_i \setminus \mathcal{R}$
	$\mathsf{plus}(\mathsf{succ}(x), y) \equiv \mathsf{succ}(\mathcal{O})$	
Orientieren		$\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathcal{O})$
Generieren	$\mathsf{succ}(\mathsf{plus}(x, y)) \equiv \mathsf{succ}(\mathcal{O})$	$\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathcal{O})$
Injektivität	$\mathsf{plus}(x, y) \equiv \mathcal{O}$	$\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathcal{O})$
Orientieren		$\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathcal{O}),\ \mathsf{plus}(x, y) \to \mathcal{O}$
Generieren	$y \equiv \mathcal{O}$	$\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathcal{O}),\ \mathsf{plus}(x, y) \to \mathcal{O}$
Inkonsistenz	"*False*"	

Im letzten Schritt hätte man stattdessen auch das kritische Paar bilden können, das sich bei der Überlappung der zweiten Regel $\mathsf{plus}(\mathsf{succ}(x), y) \to \mathsf{succ}(\mathsf{plus}(x, y))$ von $\mathcal{R}$ mit der neuen Regel $\mathsf{plus}(x, y) \to \mathcal{O}$ ergibt. Dies hätte zu der neuen Gleichung $\mathsf{succ}(\mathsf{plus}(x, y)) \equiv \mathcal{O}$ geführt, wodurch die Inkonsistenz-Regel ebenfalls anwendbar gewesen wäre.

Das folgende Lemma zeigt, dass die Transformationsregeln des Induktionsbeweiskalküls die Menge der gültigen Grundgleichungen nicht verändern, sofern die Ursprungsmenge konsistent war und das Definitionsprinzip erfüllte. Außerdem bleibt auch das Definitionsprinzip erhalten: Die Modifikation der "Orientieren"-Regel stellt sicher, dass Bedingung (a) des Definitionsprinzips stets erfüllt bleibt. Bedingung (b) bleibt insofern erfüllt, dass für jedes Paar

$(\mathcal{E}, \mathcal{R})$ während der Vervollständigung die folgende Aussage gilt: Für jeden Grundterm t existiert ein Konstruktorgrundterm q mit $t \leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} q$.

Lemma 6.3.12 (Eigenschaften von $\vdash_I$) *Sei $(\mathcal{E}, \mathcal{R}) \vdash_I (\mathcal{E}', \mathcal{R}')$ unter Verwendung der Reduktionsordnung $\succ$ und der Menge Σ^c von Konstruktoren.*

(a) Es gilt $\leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} \subseteq \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'}$.

(b) Falls für alle Grundterme t ein Konstruktorgrundterm q mit $t \leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} q$ existiert und es keine Konstruktorgrundterme $q_1 \neq q_2$ mit $q_1 \leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} q_2$ gibt, dann gilt für alle Grundterme s, t mit $s \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'} t$ auch $s \leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} t$.

(c) Falls $l \notin \mathcal{T}(\Sigma^c, \mathcal{V})$ für alle $l \to r \in \mathcal{R}$ gilt, so gilt auch $l \notin \mathcal{T}(\Sigma^c, \mathcal{V})$ für alle $l \to r \in \mathcal{R}'$.

Beweis.

(a) Alle Transformationsregeln bis auf die Injektivität verändern die Relation $\leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}}$ nicht, vgl. Lemma 6.2.6. Bei der Injektivität gilt offensichtlich $\leftrightarrow^*_{\mathcal{E}\cup\{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n)\}} \subseteq \leftrightarrow^*_{\mathcal{E}\cup\{s_1\equiv t_1,\ldots,s_n\equiv t_n\}}$.

(b) Wiederum müssen wir nur die Injektivitätsregel betrachten, da bei den anderen Transformationsregeln $\leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}} = \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'}$ gilt. Hierzu zeigen wir, dass unter den Voraussetzungen in (b) $s_i\sigma \leftrightarrow^*_{\mathcal{E}\cup\{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n)\}} t_i\sigma$ für alle i und alle Substitutionen σ gilt, die die Variablen in den s_i und t_i mit Grundtermen instantiieren. Diese Aussage genügt für die Äquivalenz der beiden Relationen $\leftrightarrow^*_{\mathcal{E}\cup\{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n)\}}$ und $\leftrightarrow^*_{\mathcal{E}\cup\{s_1\equiv t_1,\ldots,s_n\equiv t_n\}}$ auf Grundtermen.

Nach Voraussetzung gibt es zu jedem $s_i\sigma$ und $t_i\sigma$ Konstruktorgrundterme s'_i und t'_i, so dass $s_i\sigma \leftrightarrow^*_{\mathcal{E}\cup\{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n)\}} s'_i$ und $t_i\sigma \leftrightarrow^*_{\mathcal{E}\cup\{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n)\}} t'_i$ gilt. Falls es ein i mit $s'_i \neq t'_i$ gäbe, so wäre $\mathcal{E} \cup \{c(s_1,\ldots,s_n) \equiv c(t_1,\ldots,t_n)\}$ nicht konsistent, da daraus die Gleichheit der unterschiedlichen Konstruktorgrundterme $c(s'_1,\ldots,s'_n)$ und $c(t'_1,\ldots,t'_n)$ folgt. Also sind s'_i und t'_i jeweis syntaktisch gleich und somit folgt $s_i\sigma \leftrightarrow^*_{\mathcal{E}\cup\{c(s_1,\ldots,s_n)\equiv c(t_1,\ldots,t_n)\}} t_i\sigma$.

(c) Die Behauptung folgt sofort aus der geänderten "Orientieren"-Regel. □

Nun können wir die Korrektheit des Induktionsbeweisverfahrens zeigen.

Satz 6.3.13 (Korrektheit des Induktionsbeweisverfahrens) *Sei $\mathcal{E}$ ein Gleichungssystem über Σ und $\mathcal{V}$, sei $\mathcal{R}$ ein zu $\mathcal{E}$ äquivalentes konvergentes TES, das das Definitionsprinzip für $\Sigma^c \subseteq \Sigma$ erfüllt, seien $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$ und sei $\succ$ eine Reduktionsordnung. Sei $(\{s \equiv t\}, \mathcal{R}) \vdash_I \ldots$ eine faire Ableitung mit den in Def. 6.3.9 beschriebenen Transformationsregeln. Bricht die Ableitung mit "False" ab, so gilt $\mathcal{E} \not\models_I s \equiv t$. Terminiert die Ableitung mit $(\varnothing, \mathcal{R}_\omega)$, so gilt $\mathcal{E} \models_I s \equiv t$.*

Beweis. Wir betrachten zunächst den Fall $\mathcal{E} \not\models_I s \equiv t$. Dann gibt es nach Satz 6.3.5 und 6.3.8 verschiedene Konstruktorgrundterme q_1 und q_2 mit $\mathcal{E} \cup \{s \equiv t\} \models q_1 \equiv q_2$ bzw. $q_1 \leftrightarrow^*_{\mathcal{E}\cup\{s\equiv t\}} q_2$. Würde die Ableitung mit $(\varnothing, \mathcal{R}_\omega)$ terminieren, so gälte nach Lemma 6.3.12 (a) auch $q_1 \leftrightarrow^*_{\mathcal{R}_\omega} q_2$. Da die Ableitung fair ist, ist $\mathcal{R}_\omega$ nach Satz 6.2.11 ein konvergentes TES. Somit müssten q_1 und q_2 wegen der Church-Rosser Eigenschaft zusammenführbar sein. Aufgrund Lemma 6.3.12 (c) sind q_1 und q_2 aber Normalformen von $\mathcal{R}_\omega$ und somit folgt $q_1 = q_2$ im Widerspruch zur Verschiedenheit der beiden Terme. Damit ist gezeigt, dass bei einer Terminierung der Ableitung mit $(\varnothing, \mathcal{R}_\omega)$ tatsächlich $\mathcal{E} \models_I s \equiv t$ gilt.

Nun betrachten wir den Fall $\mathcal{E} \models_I s \equiv t$. Da $\mathcal{R}$ das Definitionsprinzip erfüllt, existiert nach Lemma 6.3.12 (a) für alle $(\mathcal{E}', \mathcal{R}')$ im Verlauf der Ableitung zu jedem Grundterm u ein Konstruktorgrundterm q mit $u \leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'} q$. Da es nach Satz 6.3.5 und 6.3.8 keine Konstruktorgrundterme $q_1 \neq q_2$ mit $q_1 \leftrightarrow^*_{\mathcal{E}\cup\{s\equiv t\}} q_2$ gibt (und daher auch nicht mit $q_1 \leftrightarrow^*_{\mathcal{R}\cup\{s\equiv t\}} q_2$), bleibt diese Eigenschaft nach Lemma 6.3.12 (b) während der Ableitung erhalten. Somit kann die Ableitung niemals mit "*False*" enden. □

Literaturverzeichnis

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133-178, 2000.

[Ave95] J. Avenhaus. *Reduktionssysteme*. Springer, 1995.

[BA95] F. Baader and C. A. Albayarak. *Termersetzungssysteme*. Skript zur Vorlesung, RWTH Aachen, Aachener Beiträge zur Informatik, Band 12, 1995.

[BN98] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[Bac91] L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991.

[Bir35] G. Birkhoff. On the structure of abstract algebras. *Proc. Cambridge Philophical Society*, 31:433-454, 1935.

[BR95] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14:189-235, 1995.

[Bün98] R. Bündgen. *Termersetzungssysteme.* Vieweg, 1998.

[BG99] J. Brauburger and J. Giesl. Approximating the domains of functional and imperative programs. *Science of Computer Programming*, 35:113-136, 1999.

[CR36] A. Church and J. B. Rosser. Some properties of conversion. *Trans. AMS*, 39:472-482, 1936.

[CLM08] M. Codish, V. Lagoon, and P. Stuckey. Solving Partial Order Constraints for LPO Termination. *Journal on Satisfiability, Boolean Modeling and Computation*, 5:193-215, 2008.

[DM79] N. Dershowitz and Z. Manna. Proving Termination with Multiset Orderings. *Communications of the ACM*, 22(8): 465-476, 1979.

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279-301, 1982.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69-116, 1987.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, J. van Leeuwen (ed.), pages 243–320. North-Holland, 1990.

[DF85] D. Detlefs and R. Forgaard. A procedure for automatically proving the termination of a set of rewrite rules. In *Proceedings of the First International Conference on Rewriting Techniques and Applications, RTA-85*, Dijon, France, Lecture Notes in Computer Science, 1985.

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758-771, 1980.

[Gie95] J. Giesl. Generating polynomial orderings for termination proofs. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications, RTA-95*, Kaiserslautern, Germany, Lecture Notes in Computer Science 914, pages 426-431, 1995.

[Gie03] J. Giesl. *Automatisierte Programmverifikation*. Skript zur Vorlesung, RWTH Aachen, 2003.

[Gie05] J. Giesl. *Grundlagen der Funktionalen Programmierung*. Skript zur Vorlesung, RWTH Aachen, 2005.

[Gie06] J. Giesl. *Logikprogrammierung*. Skript zur Vorlesung, RWTH Aachen, 2010.

[Her71] G. T. Herman. Strong computability and variants of the uniform halting problem. *Z. Math. Logik Grundl. Math.*, 17:115-131, 1971.

[HL78] G. Huet and J.-J. Lévy. On the uniform halting problem for term rewriting systems. Technical Report 283, INRIA, 1978.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797-821, 1980.

[HH82] G. Huet and J.-M. Hullot. Proof by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239-266, 1982.

[KL80] S. Kamin and J.-J. Lévy. Two generalizations of recursive path orderings. Unpublished note, Dept. of Computer Science, University of Illinois, Urbana, IL, USA, 1980.

[KM87] D. Kapur and D. R. Musser. Proof by consistency. *Artificial Intelligence*, 31:125–158, 1987.

[KZ95] D. Kapur and H. Zhang. An overview of rewrite rule laboratory (RRL). *Journal of Computer and Mathematics with Applications*, 29:91–114, 1995.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In *Computational Problems in Abstract Algebra*, J. Leech (ed.), pp. 263-297, Pergamon Press, 1970.

[KN85] M. S. Krishnamoorthy and P. Narendran. On recursive path ordering. *Theoretical Computer Science*, 40:323-328, 1985.

[Kru60] J. B. Kruskal. Well-quasi-orderings, the tree theorem, and Vazsonyi's conjecture. *Trans. Amer. Math. Society*, 95:21-225, 1960.

[Lan79] D. S. Lankford. On proving term rewriting systems are noetherian. Technical Report Memo MTP-3, Mathematic Department, Louisiana Technical University, Ruston, LA, 1979.

[MN70] Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proceedings of the 3rd Hawaii International Conference on System Science*, pp. 789-792, 1970.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258-282, 1982.

[MG95] A. Middeldorp and B. Gramlich. Simple termination is difficult. *Applicable Algebra in Engineering, Communication and Computing*, 6:115-128, 1995.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245-257, 1979.

[NO80] G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356-364, 1980.

[New42] M. H. A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223-242, 1942.

[Ohl02] E. Ohlebusch, *Advanced Topics in Term Rewriting*. Springer, 2002.

[Opp80] D. C. Oppen. Reasoning about recursively defined data structures. *Journal of the ACM*, 27(3):403-411, 1980.

[Rob65] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12:23-41, 1965.

[RS02] H. Rueß and N. Shankar. Deconstructing Shostak. *Proceedings of the IEEE Symposium on Logic in Computer Science, LICS 2001*, Boston, Massachusetts, 2001.

[Sho78] R. E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583-585, 1978.

[Sho84] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1-12, 1984.

[Ste95] J. Steinbach. Simplification orderings: History of results. *Fundamenta Informaticae*, 24:47-87, 1995.

[Ter03] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

[Wal95] C. Walther. *Reduktionssysteme*. Skript zur Vorlesung, TU Darmstadt, 1995.

[Zha92] H. Zhang. Implementing contextual rewriting. *Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems, CTRS-92*, Lecture Notes in Computer Science 656, pages 363-377, 1992.

Index