

Einführung in die Informatik I
Termersetzungssysteme

Prof. Bernd Brügge, Ph.D
Technische Universität München

Wintersemester 2000/2001

21 November 2000

Überblick über den Vorlesungsblock

- ❖ Termersetzungssysteme
- ❖ Terme mit freien Identifikatoren
- ❖ Substitutionsregel
- ❖ Termersetzungsregeln
- ❖ Definition Termersetzungssystem
- ❖ Termersetzungssysteme sind auch Algorithmen
 - Termersetzungssystem vs Textersetzungssystem
- ❖ Korrektheit von Termersetzungssystemen
 - Partielle Korrektheit
 - Totale Korrektheit

Wo stehen wir?

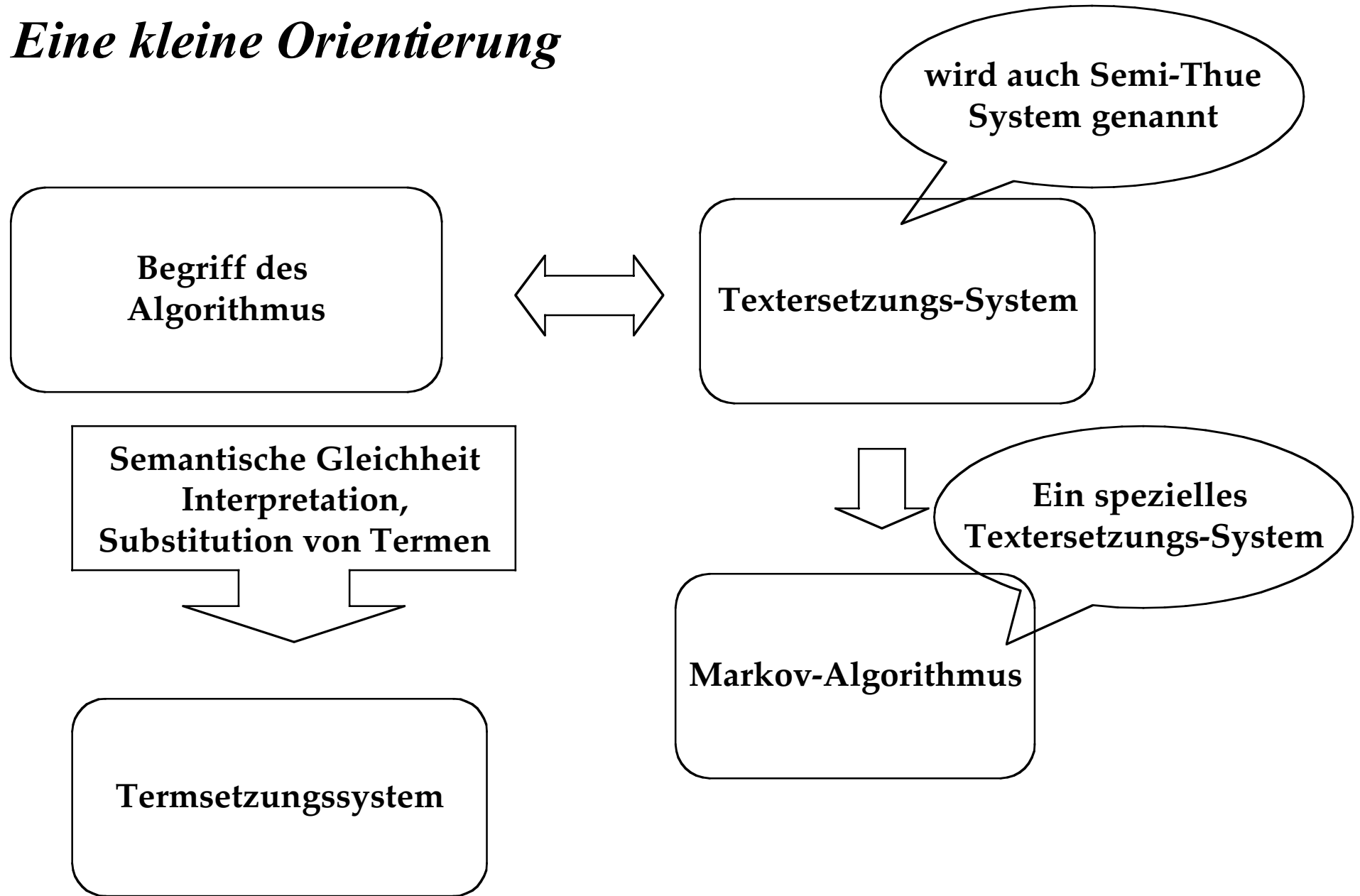
- ❖ **Oktober/November:** Modellierung, Informatik Systeme, Textersetzungs-system, Markov-Algorithmen, Algebra, Boolesche Algebra, Aussagenlogik
- ❖ **Heute:** Termersetzungs-systeme
- ❖ **Rest des Semesters:** Programmierparadigmen
 - Funktionale Programmierung (Ende November, Anfang Dezember)
 - Imperative Programmierung (Dezember)
 - Objekt-Orientierte Programmierung (Januar)
 - Ereignis-basierte Programmierung (Ende Januar, Info II)
 - Regel-basierte Programmierung (Info II)

Rubrik Sonstiges

- ❖ Einführung einer englisch-sprachigen Tutorgruppe ist gesichert.
 - Zeitpunkt ab Januar 2001
 - Bitte melden Sie sich bei der Übungsleitung, falls Sie Interesse haben.

 - Email: harrer@in.tum.de

Eine kleine Orientierung



Termersetzungssystem

- ❖ Ein Termersetzungssystem beschreibt einen Algorithmus so wie ein Textersetzungssystem, allerdings in übersichtlicherer Form:
 - Aus einer gegebenen Signatur lassen sich Terme nach festen Regeln aufbauen.
 - Für die Signatur und die Terme geben wir Interpretationen an.
 - Das ganze ergibt eine Algebra, auch Rechenstruktur genannt.
 - In dieser Rechenstruktur haben wir Terme, denen wir durch die Interpretation eine Bedeutung geben.
 - Die Substitutionsregel gestattet die Umformung dieser Terme in andere, semantisch äquivalente Terme.
 - Die Interpretation des letzten Terms in einer Ableitung beschreibt das Resultat des Algorithmus
- ❖ Noch ein **wichtiger Unterschied zu Textersetzungssystemen**: Termersetzungssysteme haben Terme mit freien Identifikatoren

Terme mit Freien Identifikatoren

❖ Beispiele:

– Gleichungen mit Unbekannten in der Mathematik sind Gleichungen zwischen Termen mit Identifikatoren.

$$- ax^2 + bx + c = 0$$

– Definitionen von Funktionen:

$$- f(x) = 2x + 1$$

– In der Mathematik heissen Terme mit freien Identifikatoren auch Polynome.

❖ Für Identifikatoren in Termen können andere Terme eingesetzt werden. Die entsprechende Abbildung heißt **Substitution von Termen** mit freien Identifikatoren.

Substitutionen von Booleschen Termen

Gilt die folgende boolesche Gleichung?

$$(w \wedge k) \wedge (w \wedge z) = (w \wedge z) \wedge (w \wedge k)$$

Ja, denn sie ist ein spezieller Fall des Kommutativgesetzes

$$x \wedge y = y \wedge x$$

Wir erhalten dies, indem wir x durch den Term $(w \wedge k)$ ersetzen, und y durch den Term $(w \wedge z)$. Wir schreiben dies folgendermaßen:

$$[(w \wedge k)/x, (w \wedge z)/y]$$

und lesen dann:

Substitution von x durch $(w \wedge k)$ und y durch $(w \wedge z)$.

Eine andere Schreibweise benutzt den $:=$ Operator:

$$[x := (w \wedge k), y := (w \wedge z)]$$

Substitution von Termen

❖ **Definition Substitution in Termen mit freien Identifikatoren:**

Seien t_1 und t_2 Boolesche Terme mit freien Identifikatoren ID und sei $x \in ID$ ein Identifikator. Dann bezeichnen wir mit

$t_1[t_2/x]$ den Term, den wir aus t_1 erhalten, indem wir den Identifikator x an allen Stellen in t_1 durch den Term t_2 ersetzen. (*Broy und Goos benutzen diese Notation, Gries benutzt $t_1[x:=t_2]$*)

Beispiel:

$x \wedge (\neg y) [\mathbf{False}/x]$ bedeutet: Ersetze alle x durch **False**.

Resultat: **False** \wedge ($\neg y$)

❖ Wir können die Substitution zusammen mit der Gleichheitsregel bei der Ableitung von Booleschen Formeln in der Aussagenlogik kombinieren:

$$\neg x \wedge (\neg y) [\mathbf{False}/x] = \mathbf{False} \wedge (\neg y)$$

Instanz eines Terms

- ❖ Sei t ein Term mit freien Identifikatoren. Ein Term r heißt **Instanz** des Terms t , wenn r durch Substitution von Identifikatoren aus t entsteht.
 - Beispiel: Gegeben sei der Term $t = \text{mult}(\text{add}(\text{succ}(x), y), z)$, wobei x , y und z freie Identifikatoren sind.
Wir benutzen die Substitution
 - $t[\text{zero}/x, \text{succ}(\text{zero})/y, \text{succ}(\text{succ}(\text{zero}))/z]$und erhalten die folgende Instanz von t
 - $\text{mult}(\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})), \text{succ}(\text{succ}(\text{zero})))$
- ❖ Analog zu Booleschen Termen lassen sich ganz allgemein alle Terme mit freien Identifikatoren interpretieren, und zwar durch Belegungen (siehe verallgemeinerte Definition der Belegung und Interpretation in 06_Aussagenlogik, Folie 49).

Beispiel einer Interpretation: Erste Idee

- ❖ Gegeben seien die Menge aller wohldefinierten Terme über der Algebra der natürlichen Zahlen, d.h. der Signatur $\Sigma = \{\text{zero}, \text{succ}, \text{pred}, \text{add}, \text{mult}, \text{div}\}$, den "bekanntem" konkreten Funktionen $\{0, +, -, *, /\}$ und die Trägermenge \mathbb{N} .
- ❖ Wir definieren dann folgende Interpretation (Erste Version):

$$- I[\text{zero}] = 0$$

$$- I[\text{succ}(x)] = x + 1$$

$$- I[\text{pred}(x)] = x - 1$$

$$- I[\text{add}(x, y)] = x + y$$

$$- I[\text{mult}(x, y)] = x * y$$

$$- I[\text{div}(x, y)] = x / y$$

Beispiel einer Interpretation: Zweite Version, wesentlich formaler! Mit Fallunterscheidungen

- ❖ Gegeben seien die Menge aller wohldefinierten Terme über der Algebra der natürlichen Zahlen, d.h. der Signatur $\Sigma = \{\text{zero}, \text{succ}, \text{pred}, \text{add}, \text{mult}, \text{div}\}$, den "bekanntem" konkreten Funktionen $\{0, +, -, *, /\}$ und die Trägermenge \mathbb{N} .
- ❖ Wir definieren dann folgende Interpretation:

$$- I[\text{zero}] = 0$$

$$- I[\text{succ}(x)] = I[x] + 1$$

$$- I[\text{pred}(x)] = \begin{cases} I[x] - 1, & \text{falls } I[x] > 0 \\ \perp \text{ (undefiniert),} & \text{sonst} \end{cases}$$

$$- I[\text{add}(x, y)] = I[x] + I[y]$$

$$- I[\text{mult}(x, y)] = I[x] * I[y]$$

$$- I[\text{div}(x, y)] = \begin{cases} I[x] / I[y], & \text{falls } I[y] > 0 \\ \perp, & \text{sonst} \end{cases}$$

Beispiel von Interpretation von Grundtermen

- ❖ Zu einer gegebenen Signatur Σ einer Rechenstruktur existiert immer die Menge aller syntaktisch korrekten Terme, die nur über Elementen aus Σ gebildet werden, auch Grundterme genannt.
- ❖ Die Menge aller Grundterme über den natürlichen Zahlen enthält z.B.
 - $\text{succ}(\text{zero})$
 - oder
 - $\text{mult}(\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})), \text{succ}(\text{succ}(\text{zero})))$
- ❖ Mit der soeben definierten Interpretation erhalten wir für diese Grundterme folgende Werte:
 - $I[\text{succ}(\text{zero})] = 1$
 - $I[\text{mult}(\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})), \text{succ}(\text{succ}(\text{zero})))] = 4$
- ❖ Warum erhalten wir das Resultat 4?

Definition: Interpretation

Für jede Belegung β ist die Interpretation $I_\beta[t]$ eines Terms t mit freien Variablen (Identifikatoren) aus X für eine Rechenstruktur A :

- ❖ Für alle Elemente $x \in X : I_\beta[x] = \beta(x)$
- ❖ Für alle $f \in \Sigma$ mit den zugeordneten konkreten Funktionen f^k :
 - $I_\beta[f(t_1, \dots, t_n)] = f^k(I_\beta[t_1], \dots, I_\beta[t_n])$

Interpretation des Beispiel-Grundterms

$$I_\beta[\text{mult}(\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})), \text{succ}(\text{succ}(\text{zero})))] =$$

-- Anwendung von $I_\beta[\text{mult}(x,y)] = I_\beta[x] * I_\beta[y]$

$$I_\beta[\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero}))] * I_\beta[\text{succ}(\text{succ}(\text{zero}))] =$$

-- Anwendung von $I_\beta[\text{add}(x,y)] = I_\beta[x] + I_\beta[y]$, und $I_\beta[\text{succ}(x)] = I_\beta[x] + 1$

$$(I_\beta[\text{succ}(\text{zero})] + I_\beta[\text{succ}(\text{zero})]) * (I_\beta[\text{succ}(\text{zero})] + 1) =$$

-- Mehrfache Anwendung von $I_\beta[\text{succ}(\text{zero})] = I_\beta[\text{zero}] + 1$

$$(I_\beta[\text{zero}] + 1 + I_\beta[\text{zero}] + 1) * (I_\beta[\text{zero}] + 1 + 1) =$$

-- Mehrfache Anwendung von $I_\beta[\text{zero}] = 0$

$$(0 + 1 + 0 + 1) * (0 + 1 + 1) =$$

-- Anwendung der konkreten Funktionen $+$ und $*$ für die gegebenen Argumente

4

Termersetzungsregeln

- ❖ **Definition Termersetzungsregel:** Sei eine Familie X von Identifikatoren (Variablen) gegeben. Ein Paar (t, r) von Termen t, r gleicher Sorte mit freien Identifikatoren aus X heißt eine Termersetzungsregel.
 - Für die Regel schreiben wir wieder $t \rightarrow r$
- ❖ Eine Termersetzungsregel bezeichnet in Wirklichkeit eine Menge von Regeln, weil sie freie Identifikatoren enthält:
 - Ersetzen wir einige oder alle Identifikatoren in der Regel, so erhalten wir eine Instanz der Regel.
- ❖ **Definition Instanz einer Regel:**
 - $t[t_1/x_1, \dots, t_n/x_n] \rightarrow r[t_1/x_1, \dots, t_n/x_n]$ ist eine Instanz von $t \rightarrow r$
- ❖ **Beispiel:**
 - Die Regel $\text{pred}(\text{succ}(x)) \rightarrow x$ mit Substitution $[\text{zero}/x]$ ergibt die Instanz $\text{pred}(\text{succ}(\text{zero})) \rightarrow \text{zero}$

Termersetzungsschritt

- ❖ Aus einer Regel wird ein Termersetzungsschritt gewonnen, indem wir die Instanz einer Regel auf einen beliebigen Teilterm eines vorliegenden Terms anwenden.
- ❖ Beispiel:
 - Mit der Instanz **pred(succ(zero))** \rightarrow **zero** der Regel **pred(succ(x))** \rightarrow **x** geht der Term **succ(succ(pred(succ(zero))))** in einem Termersetzungsschritt über in den Term **succ(succ(zero))**.
- ❖ Schreibweise:
 - $\text{succ(succ(pred(succ(zero))))} \Rightarrow \text{succ(succ(zero))}$

(Vergleiche Ableitung bei Textersetzungssystemen)

Termersetzungssystem

- ❖ **Definition Termersetzungssystem:** Eine Menge R von Termersetzungsregeln über einer Signatur Σ heißt Termersetzungssystem über Σ .
- ❖ **Definition Berechnung:** Gilt für die Folge von Termen $t_0, t_1, t_2, \dots, t_n$
 - $t_i \Rightarrow t_{i+1}$, d.h. t_{i+1} entsteht aus t_i durch einen Termersetzungsschritt mit der Instanz einer Regel aus R ,
 - so heißt die Folge $t_0, t_1, t_2, \dots, t_n$ Berechnung von R für den Term t_0 .
- ❖ **Definition Terminaler Term:** Ein Term, der nicht weiter abgeleitet werden kann.
- ❖ **Definition Terminierende Berechnung:** Eine endliche Folge $t_0, t_1, t_2, \dots, t_n$, in der t_n terminal ist.
 t_0 heißt die Eingabe, t_n heißt die Ausgabe.
- ❖ **Definition Nichtterminierende Berechnung:** Eine Berechnung, die aus einer unendlichen Folge von Termen besteht.

Ein Termersetzungssystem ist ein Algorithmus

- ❖ Sei ein Termersetzungssystem R gegeben. Dann definiert R mit den folgenden Metaregeln einen Algorithmus:
 - Enthält R eine Ersetzungsregel, die auf den Term t angewendet den Termersetzungsschritt $t \Rightarrow r$ ergibt, dann wird der Algorithmus mit dem Term r statt t fortgesetzt.
 - Enthält R keine Ersetzungsregel, die auf den Term t anwendbar ist, dann endet der Algorithmus mit t als Resultat.

Beispiel eines Algorithmus: Reduktion von Termen in Normalform

- ❖ Gegeben sei die Rechenstruktur der natürlichen Zahlen mit der Interpretation von Folie 12.
- ❖ Für diese Rechenstruktur lassen sich alle natürlichen Zahlen in der Normalform $\text{succ}(\dots(\text{succ}(\text{zero})))$ darstellen.
- ❖ Wir konstruieren einen Algorithmus für die Konvertierung von Termen (die kein div enthalten) in die Normalform, indem wir ein Termersetzungssystem mit folgenden Regeln definieren.

$$\begin{aligned} R = \{ & \\ & \text{pred}(\text{succ}(x)) \quad \rightarrow x \\ & \text{add}(x, \text{succ}(y)) \quad \rightarrow \text{succ}(\text{add}(x,y)) \\ & \text{add}(x, \text{zero}) \quad \rightarrow x \\ & \text{mult}(\text{succ}(x), y) \quad \rightarrow \text{add}(\text{mult}(x,y), y) \\ & \text{mult}(x, \text{succ}(y)) \quad \rightarrow \text{add}(\text{mult}(x,y), x) \\ & \text{mult}(\text{zero}, \text{zero}) \quad \rightarrow \text{zero} \\ & \} \end{aligned}$$

Was bedeuten die einzelnen Regeln in dem Algorithmus?

Termersetzungssystem $R = \{$

$\text{pred}(\text{succ}(x)) \rightarrow x$ --1

$\text{add}(x, \text{succ}(y)) \rightarrow \text{succ}(\text{add}(x,y))$ --2

$\text{add}(x, \text{zero}) \rightarrow x$ --3

$\text{mult}(\text{succ}(x),y) \rightarrow \text{add}(\text{mult}(x,y), y)$ --4

$\text{mult}(x, \text{succ}(y)) \rightarrow \text{add}(\text{mult}(x,y), x)$ --5

$\text{mult}(\text{zero},\text{zero}) \rightarrow \text{zero}$ --6

$\}$

❖ Regel 1 eliminiert pred

❖ Regeln 2 und 3 definieren einen Algorithmus für die Addition

❖ Regeln 4, 5 und 6 definieren einen Algorithmus für die Multiplikation

Termersetzungssystem vs Textersetzungssystem

Termersetzungssystem $R = \{$
mult(succ(x), y) \rightarrow add(mult(x,y), y)
mult(x, succ(y)) \rightarrow add(mult(x,y), x)
mult(zero, zero) \rightarrow zero
 $\}$

Textersetzungssystem $T = \{$
 $I>*< \rightarrow >*<d$
dI \rightarrow Imd
dm \rightarrow md
d> \rightarrow >
 $\diamond * < \rightarrow <e$
eI \rightarrow e
em \rightarrow Ie
e> \rightarrow > $\}$



Beide Regelsysteme beschreiben einen Algorithmus für die Multiplikation!

Termersetzungssysteme sind viel lesbarer als Textersetzungssysteme.

Beide Regelsysteme gehören zu einem bestimmten Programmierparadigma, der regelbasierten Programmierung
Insgesamt werden wir fünf verschiedene Programmierparadigmen kennenlernen.

Korrektheit von Termersetzungssystemen

❖ **Definition Partielle Korrektheit:** Eine Termersetzungsregel $t \rightarrow r$ heisst partiell korrekt in einer Rechenstruktur A , falls für jede Belegung β in A bzgl. der Interpretation I_β gilt:

$$- I_\beta[t] = I_\beta[r]$$

❖ Das bedeutet, dass die Terme t und r in der gewählten Applikationsdomäne gleich sind.

❖ Beispiel: Das soeben definierte Regelsystem $R = \{$

$$\text{pred}(\text{succ}(x)) \rightarrow x$$

$$\text{add}(\text{succ}(x), y) \rightarrow \text{succ}(\text{add}(x, y))$$

$$\text{add}(\text{zero}, y) \rightarrow y$$

$$\text{mult}(\text{succ}(x), y) \rightarrow \text{add}(y, \text{mult}(x, y))$$

$$\text{mult}(x, \text{succ}(y)) \rightarrow \text{add}(x, \text{mult}(x, y))$$

$$\text{mult}(\text{zero}, \text{zero}) \rightarrow \text{zero} \}$$

ist partiell korrekt bezüglich der Interpretation auf Folie 12

Partielle Korrektheit von R

- ❖ Um die partielle Korrektheit von R zu zeigen, muss man die partielle Korrektheit jeder Regel zeigen.
- ❖ Beispiel: Wir wollen die partielle Korrektheit für die Regel
 - $\text{add}(\text{succ}(x), y) \rightarrow \text{succ}(\text{add}(x, y))$ beweisen.Wir wenden die Definition der semantischen Äquivalenz an:
 - $I[\text{add}(\text{succ}(x), y)] = I[\text{succ}(\text{add}(x, y))]$
- ❖ Dann wenden wir die rekursive Definition der Interpretation an:
 - $I[\text{succ}(x)] + I[y] = I[\text{add}(x, y)] + 1$
 - $(I[x] + 1) + I[y] = (I[x] + I[y]) + 1$
- ❖ Dann zeigen wir für alle Belegungen β , dass die Gleichung gilt:
 - $(\beta(x) + 1) + \beta(y) = (\beta(x) + \beta(y)) + 1$
- ❖ Die Gültigkeit dieser Gleichung folgt aus der Kommutativität und Assoziativität der Addition.
- ❖ Die partielle Korrektheit der anderen Regeln von R ist analog.

Totale Korrektheit

- ❖ Die partielle Korrektheit eines Regelsystems wird bewiesen, indem man die partielle Korrektheit jeder Regel beweist.
- ❖ Totale Korrektheit lässt sich nur für das gesamte Regelsystem beweisen.
- ❖ **Definition Totale Korrektheit:** Ein Regelsystem ist total korrekt, wenn es partiell korrekt ist und terminiert.

Zusammenfassung

- ❖ Eine **Termersetzungssystem** beschreibt einen Algorithmus
- ❖ Ein Termersetzungssystem ist eine Menge von **Termersetzungsregeln**
- ❖ Die wesentliche Idee eines Termersetzungssystems ist das Konzept der **Substitution in Termen mit freien Identifikatoren**
- ❖ **Berechnung:** Eine Folge von Termen, die durch die Anwendung von Instanzen von Termersetzungsregeln entsteht.
- ❖ Termersetzungssysteme sind einfacher zu lesen als Textersetzungssysteme
- ❖ Termersetzungssysteme und Textersetzungssysteme sind Beispiele eines spezifischen Programmier-Paradigmas, der regelbasierten Programmierung.
- ❖ Es gibt viele Programmier-Paradigmen, 5 davon werden Sie in dieser Vorlesung genauer kennenlernen.

Ausblick auf die nächsten Vorlesungsblöcke

- ❖ Funktionale Programmierung
 - 27/28 November 4/5 Dezember
- ❖ Imperative Programmierung
 - 11/12, 19/20 Dezember
- ❖ Objekt-orientierte Programmierung, Ereignis-basierte Programmierung
 - 9, 16/17, 23/24, 30/31 Januar
- ❖ Ereignis-basierte Programmierung
 - weiter in Info II
- ❖ Regel-basierte Programmierung:
 - Info II