3.12.2002

1 Deterministische Textersetzungssysteme

1. Bisher: Nichtdeterministische Textersetzungssysteme

$(V^*, \{P\})$ wobei $\{P\}$ eine Menge von der Form $s \to t$, $s, t \in V^*$. Die Auswahl einer geeigneten erfolgt nichtdeterministisch. (Semi-Thue-Systeme)

1.2 Markow-Algorithmen

Gegeben sei ein Textersetzungssystem (V, P). Neben Regeln der Form $s \to t$ gibt es haltende Regeln: $s' \to_. t'$

Vorgehensweise (Meta-Regel)

- Die Regeln sind in einer bestimmten Reihenfolge gegeben
- Wähle in jedem Berechnungsschr. eine geeignete Regel
- Ist eine Regel auf mehrere Teilworte anwendbar, so wende man die Regel auf das am weitesten links stehende Teilwort an
- Wende Regeln solange an, bis eine haltende Regel angewandt wurde oder keine Regel mehr anwendbar ist.

Bemerkung: Erlaubt man im Eingabetext zusätzliche Zeichen, $\alpha, \beta, \ldots$, so läßt mit Hilfe von Markow-Alg. jede bel. Berechn. durchführen.

Beispiel: (Addition von 1 auf Dualzahl)

$$V = \{0, L, \alpha, \beta\} \cup E$$

$$P = \{$$

$\alpha L \to L\alpha$	(1)	
$\alpha 0 \to 0\alpha$	(2)	
$\alpha \to \beta$	(3)	
$L\beta \to \beta 0$	(4)	
$0\beta \to .L$	(5)	haltende Regeln
$\beta \to .L$	(6)	
$\varepsilon \to \alpha$	(7)	ε-Regel

$$\}$$

Eingabe: $L0LL$ ($\hat{=}$ 11)

$$\varepsilon \cdot L0LL \overset{(7)}{\Longrightarrow} \alpha L0LL \overset{(1)+(2)}{\Rightarrow} \ldots \Rightarrow L0LL\alpha$$

$$\overset{(3)}{\Rightarrow} L0LL\beta \overset{(4)}{\Rightarrow} L0L\beta 0 \overset{(4)}{\Rightarrow} L0\beta 00$$

$$\overset{(5)}{\Rightarrow} LL00$$

Bemerkung: Semi-Thue- und Markov-Systeme sind formale Systeme:

Das Tupel $(U, \Rightarrow)$ ist ein formales System, wenn

- U endlich oder abzählbar ∞ ist
- $\Rightarrow \subseteq U \times U$, eine Relation
- $\exists$ Algorithmus, der jedes $t \in U$

mit $L \Rightarrow F$ in endlich vielen Schritten ableitet.

Ein formales System mit (endlich vielen) Metaregeln nennt man ein Kalkül

2. Rechenstrukturen, Signaturen und Termersetzungssysteme

In der Informatik arbeiten wir häufig mit Rechenstrukturen, die sich aus Daten (Trägermengen) und Funktionen (Operationen zusammensetzen

2.1 Signatur:

Definition: Eine Signatur Σ ist ein Paar (S, F) von Mengen S und F, wobei

- S die Menge der Sorten, d.h. der Bezeichner der Trägermengen
- F die Menge der Funktionssymbole

die Funktionalität sei dabei durch

$f : (s_1, \ldots, s_n) \to s_{n+1}$ $\quad f \in F$

gegeben

($s_1, \ldots, s_n$: Sorten der Funkt.-par; s_{n+1}: Rückgabesorte)

Bemerkung: Die Signatur definiert nur Bezeichner!

Beispiel: Signatur boolescher Werte

$S = \{bool\}$

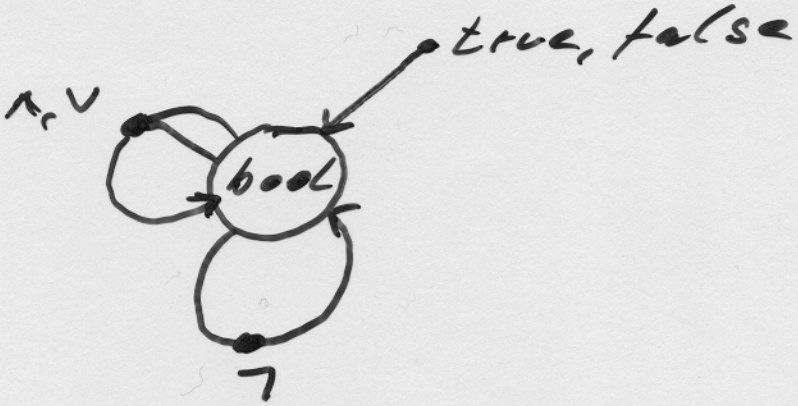
$F = \{true, false, \neg, \wedge, \vee\}$

Funktionalität:

fct true = () bool
fct false = " "
fct $\wedge$ = (bool, bool) bool
" $\vee$ = "
fct $\neg$ = (bool) bool

} Infix-operatoren

Signaturgraph zur graphischen Darstellung der Signatur



2.2 Grundterme

Gegeben sei eine Signatur $\Sigma = (S, F)$. Dann ist W_Σ die Menge der Grundterme der Sorte $s \in S$ gegeben durch

- jedes nullstellige Funktionssymbol, fct f = () s, ist ein Grundterm
- jede Zeichenreihe $f(t_1, \ldots, t_n)$ mit $f \in F$ und fct f = $(s_1, \ldots, s_n)$ s_{n+1} ist

ein Grundterm der Sorte s_{nat}

Beispiel: Wir betrachten folgende

Signatur

$S = \{ bool, nat \}$

$F = \{ true, false, \neg, \vee, \wedge, succ, pred, zero, mult, add, sub, div, =, < \}$

$fct\ zero = () nat$
$fct\ succ = (nat) nat$
$fct\ pred = \quad "$
$fct\ add = (nat, nat) nat$
$\vdots$

} Praefix-operatoren

Beispiel für Grundterme:

- succ(zero)
- succ(succ(succ(zero)) = zero)
- true $\wedge$ (succ(succ(zero)) = zero)
- $\vdots$

Bemerkung:

- Ein Term heißt atomar, wenn er nicht aus weiteren Termen zusammengesetzt ist: true, zero

2.3 Terme mit (freien) Identifikatoren

Erläuterung: Identifikatoren sind Platzhalter für Terme, die später an der entspr. Stelle eingesetzt werden.
„Frei" heißen sie, weil sie an keine best. Menge von Termen gebunden sind.

Es bezeichne W_Σ die Menge aller Grundterme von $\Sigma = (S, F)$ und $X = \{X_s : s \in S\}$ eine Menge von Mengen von Identifikatoren (jede Sorte hat ihre Menge von Id.).
Dann bezeichnet $W_{\Sigma'}$ mit
$\Sigma' = \{S, F \cup \{x \in X_s, s \in S\}\}$ die Menge der Terme mit (freien) Identif

Beispiele: $X_{bool} = \{u, v\}$, $X_{nat} = \{x, y, z\}$

Terme der Signatur aus 2.2:

$u \wedge (x = succ(y))$

$(u \vee v) \vee (zero < pred(zero))$

2.4 Substitution in Termen:

Sei t ein Term, x ein Ident. und r ein Term der Sorte s. Dann bezeichnet $t[r/x]$ (sprich: r für x), den Term t', der aus t durch Substitution von x durch r entsteht

Verallg.:
$t[r_1/x_1, \ldots, r_n/x_n]$ bezeichnet den Term t' der durch simultane Subst. von $x_1, \ldots, x_n$ durch $r_1, \ldots, r_n$ entsteht.

Beispiel:

① $t := add(succ(x), succ(succ(y)))$

$t[zero/x, succ(z)/y]$

$\Rightarrow t' = add(succ(zero), succ(succ(succ(z))))$

② Betrachte die Signatur von ~~Zeichen-~~ Listen ~~sequenzen~~ in Ocaml

$\Sigma = \{S, F\}$

$S = \{char, seq\ char\}$

$F = \{'a', \ldots, 'z', [], ::\}$

$fct\ 'a' = () char$

$\vdots$

$fct\ [] = () seq\ char$

$fct\ :: = (char, seq\ char)\ seq\ char$

Beispielterme:

$t_1 := ('a' :: ('a' :: []))$

$t_2 := ('c' :: x)$; $t_3 := y :: x$

$t_2[t_1/x] = 'c' :: 'a' :: 'a' :: []$

$(t_2[(t_3[z/x, 'c'/y])/x])[x/z]$

$\vdots$

$= 'c' :: 'c' :: x$