Theoretische Grundlagen des Software Engineering

12: Termersetzungssysteme

Stephan Schulz
schulz@eprover.org

Software Systems Engineering

Stiftung Universität Hildesheim 2003

Reduktionssysteme

Definition: Reduktionssystem

Ein Reduktionssystem ist ein Tupel

$$(A, \rightarrow)$$

Dabei gilt:

- A ist eine beliebige Menge (die Trägermenge)
- $\rightarrow \subseteq A \times A$ ist eine zweistellige Relation über A (die Reduktionsrelation)

Ausprägungen

Abstrakte Reduktionssysteme

- Universelle Grundlagen

Wortersetzungssysteme

- Berechenbarkeitstheorie/Grammatiken/Turing- Maschinen

Termersetzungssysteme

- Theorembeweisen, Programmsynthese, Funktionale Programmiersprachen

Polynom-Rewriting

- Computer-Algebra (Buchbergers Algorithmus/Gröbner-Basen)

Graph- Rewriting

- Verallgemeinerung von Termersetzung, Modellbasierte Entwicklung

Termersetzungssysteme

Termersetzungssysteme

Idee

Reduktionssystem $(A, \rightarrow)$

A:

Menge von Termen

$\rightarrow$:

Endlich beschrieben durch

Regeln

und

Gleichungen

Beispiel

Betrachte folgendes Regelwerk:

1. $add(x, 0) \rightarrow x$
2. $add(x, s(y)) \rightarrow s(add(x, y))$

Idee: Spezifiziere Addition

- 0 repräsentiert $0 \in \mathbb{N}$
- s ist die Nachfolgerfunktion ("$+1$")
- $s^i(0)$ repräsentiert $i \in \mathbb{N}$

Beispiel (fortgesetzt)

(1) $add(x,0) \rightarrow x$ **(2)** $add(x,s(y)) \rightarrow s(add(x,y))$

$$
\begin{array}{rc}
 & add(s(s(0)), add(s(0), s(s(0)))) \\
(1) \rightarrow & add(s(s(0)), s(add(s(0), s(0)))) \\
(1) \rightarrow & add(s(s(0)), s(s(add(s(0), 0)))) \\
(2) \rightarrow & add(s(s(0)), s(s(s(0)))) \\
(1) \rightarrow & s(add(s(s(0)), s(s(0)))) \\
(1) \rightarrow & s(s(add(s(s(0)), s(0)))) \\
(1) \rightarrow & s(s(s(add(s(s(0)), 0)))) \\
(2) \rightarrow & s(s(s(s(s(0)))))
\end{array}
$$

Beispiel (fortgesetzt)

(1) $add(x,0) \to x$ **(2)** $add(x,s(y)) \to s(add(x,y))$

$$
\begin{aligned}
& add(s(s(0)), s(add(s(0), s(0)))) \\
\to\ & add(s(s(0)), s(s(add(s(0), 0)))) \\
& \xrightarrow{*} s(s(s(s(s(0))))) \\
& \text{vs.} \\
& add(s(s(0)), s(add(s(0), s(0)))) \\
\to\ & s(add(s(s(0)), add(s(0), s(0)))) \\
\to\ & s(add(s(s(0)), s(add(s(0), 0)))) \\
\to\ & s(s(add(s(s(0)), add(s(0), 0)))) \\
\to\ & s(s(add(s(s(0)), s(0)))) \\
\to\ & s(s(s(add(s(s(0)), 0)))) \\
\to\ & s(s(s(s(s(0)))))
\end{aligned}
$$

Variablen und Funktionssymbole

Definition: Variablenmenge V

V ist eine abzählbare Menge (von Variablensymbolen)

Definition: Signatur Σ

Eine Signatur Σ ist eine Menge von Funktionssymbolen mit assoziierter Stelligkeit

Strikt analog zur Prädikatenlogik,
nur ohne Prädikatssymbole

Variablen und Funktionssymbole

Anmerkungen

- Wir verlangen, dass V, Σ disjunkt sind
- Funktionssymbole haben ein Stelligkeit $n \geq 0$
  - Schreibweise: $f|_n$ oder $arity(f) = n$
  - Oft implizit ersichtlich aus der Verwendung
- Funktionsssymbole mit Stelligkeit $n = 0$ heißen Konstanten
- Konvention für die Vorlesung:
  - Variablen: $x, y, z, x_1, y_s, \ldots$
  - Konstanten: $a, b, c, d, \ldots$
  - Funktionssymbole: $f, g, h \ldots$

Terme

Definition

Gegeben: Σ, V

$\mathit{Term}(\Sigma, V)$ ist die kleinste Menge mit:

$V \subseteq \mathit{Term}(\Sigma, V)$

Wenn
- $f \in \Sigma$ mit Stelligkeit n
- $t_1, \ldots, t_n \in \mathit{Term}(\Sigma, V)$

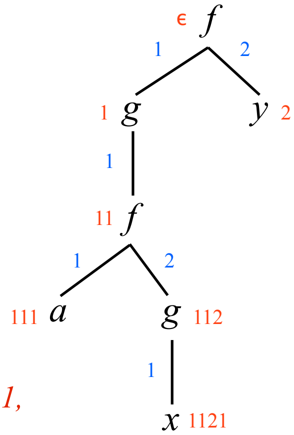
dann

$f(t_1, \ldots, t_n) \in \mathit{Term}(\Sigma)$

$\mathit{Term}(\Sigma, V)$ ist die Menge der Terme über Σ und V

Terme als Bäume

$f(g(f(a,g(x))),y) =$



Pos(t)={ϵ, 1, 2, 11,111, 112, 1121}

Positionen

Definition: Positionen in Termen

Positionen (oder Stellen) sind Worte über $\mathbb{N}$ und bezeichnen Teilterme eines Terms. Sei $t \in Term(F, V)$. Die Menge $Pos(t)$ ist rekursiv definiert:

- $Pos(t) = \{\epsilon\}$ falls $t \in V$
- Ansonsten gilt: $t = f(t_1, \ldots, ts_n)$ für $f|_n \in F$ und $t_1, \ldots, t_n \in Term(F, V)$. Dann ist

$$Pos(s) = \{\epsilon\} \cup \bigcup_{1 \leq i \leq n} Pos(t_i)$$

Teilterme

Definition: Teilterme und Positionen

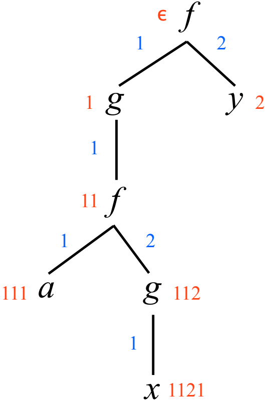
Sei $t \in Term(\Sigma, V)$ und $p \in Pos(t)$. Der Teilterm von t an Stelle p, $t|_p$ ist rekursiv definiert wie folgt:

- Fall 1: $p = \epsilon$. Dann ist $t|_p = t$
- Fall 2: $p = iq$. Dann hat t die Form $f(t_1, \ldots, t_m)$ und es gilt $t_{iq} = t_i|q$

s ist ein Teilterm von t, falls $\exists p \in Pos(t) : s = t_p$

s ist ein echter Teilterm von t, falls $\exists p \in Pos(t) : p \neq \epsilon$ und $s = t_p$

Teilterme und Positionen



$$
\begin{aligned}
t &= f(g(f(a,g(x))),y)\\
t|_2 &= y\\
t|_{112} &= g(f(a,g(x)))|_{12}\\
&= f(a,(g(x))|_2\\
&= g(x)\\
t|_{11} &= f(a,g(x))
\end{aligned}
$$

Termersetzung

Definition

Seien $s, t \in Term(\Sigma, V)$ und $p \in Pos(t)$

$t[s]_P$ ist der der Term, der entsteht, wenn $t|_p$ in t durch s ersetzt wird.

Formal: Induktive Definition über $|p|$

- Fall 1 ($p = \epsilon$):
  Dann gilt $t[s]_\epsilon = s$
- Fall 2 ($p = iq$ und $t = f(t_1, \ldots, t_n)$):
  Dann gilt: $f(t_1, \ldots, t_n)[s]_{iq} = f(t_i, \ldots, t_i[s]_q, \ldots t_n)$

Termersetzung

Beispiele

Sei $t = f(g(f(a, g(x))), y)$

- $t[b]_{11} = f(g(b), y)$
- $t[g(g(a))]_2 = f(g(f(a, g(x))), g(g(a)))$
- $t[f(c, c)]_\epsilon = f(c, c)$

Eigenschaften von Positionen

Lemma (Pos- 1)

Seien $s, t, r \in Term(\Sigma, V)$ und seien $p, q \in \mathbb{N}^*$. Dann gilt:

(a) : Wenn $pq \in Pos(t)$, dann $p \in Pos(t)$

(b) : Wenn $pq \in Pos(t)$, dann $(t|_p)|_q = t|_{pq}$

Variablenmenge

Definition: Variablenmenge eines Terms

Sei $t \in Term(\Sigma, V)$. Die Variablenmenge von t ist die Menge der in t vorkommenden Variablen, also:

$$Var(t) = \{t|_p | p \in Pos(t)\} \cap V$$

Beispiele:

- $Var(x) = \{x\}$
- $Var(f(g(a), b)) = \emptyset$
- $Var(f(g(x), y)) = \{x, y\}$

Grundterme

Definition

Die Menge der Grundterme über Σ ist $\mathit{Term}(\Sigma, \emptyset)$

Schreibweise: $\mathit{Term}(\Sigma)$

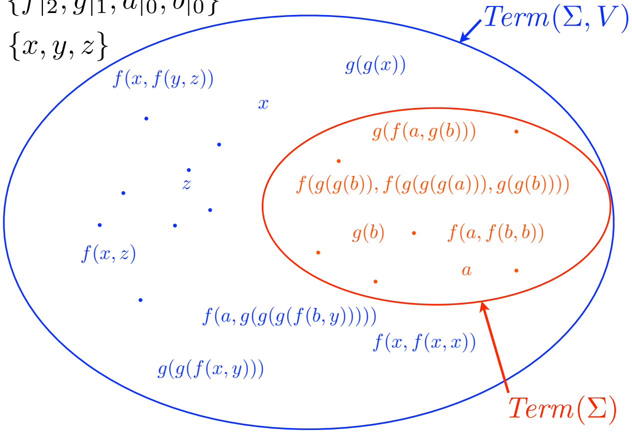
Es gilt: $\mathit{Term}(\Sigma) = \{t \in \mathit{Term}(\Sigma, V) | \mathit{Var}(t) = \emptyset\}$

Der Begriff grund wird mit gleicher Bedeutung (“variablenfrei”) auf auf Tupel, Regeln, Gleichungen, und Mengen von Termen angewendet!

Terme und Grundterme

$\Sigma = \{f|_2, g|_1, a|_0, b|_0\}$

$V = \{x, y, z\}$



Regeln und Gleichungen

Definition

Seien Σ und V gegeben.
Eine Gleichung (über $Term(\Sigma, V)$) ist ein Tupel $(s, t) \in Term(\Sigma, V) \times Term(\Sigma, V)$.

- Wir schreiben die Gleichung (s, t) als $s \simeq t$
- Gleichungen sind ungerichtet, d.h. $s \simeq t$ ist implizit äquivalent to $t \simeq s$

Eine Regel (über $Term(\Sigma, V)$) ist ein Tupel $(s, t) \in Term(\Sigma, V) \times Term(\Sigma, V)$.

- Wir schreiben die Regel (s, t) als $s \rightarrow t$
- Regeln sind gerichtet!
- Die Seiten einer Regel heißen linke Seite (left hand side, LHS) und rechte Seite (right hand side RHS)

Substitutionen (1)

Definition

Seien Σ und V gegeben. Eine *Substitution* ist eine Funktion $\sigma : V \rightarrow Term(\Sigma, V)$ mit der Eigenschaft, dass $\{x | \sigma(x) \neq x\}$ endlich ist.

- $Dom(\sigma) = \{x | \sigma(x) \neq x\}$ heißt die Domäne von σ
- Anschaulich: Eine Substitution ersetzt (endlich viele) Variablen durch Terme
- Beachte: $V \subseteq Term(\Sigma, V)$ - Substitutionen können also auch Variablen einsetzen
- Schreibweise: $\sigma = \{x \leftarrow t_1, y \leftarrow t_2\}$
- Aus unklaren Gründen findet man in der Literatur oft $x\sigma$ statt $\sigma(x)$

Substitutionen (2)

Substitutionen können auch auf Terme, Gleichungen und Regeln angewendet werden:

Definition

Eine Substitution σ wird rekursiv fortgesetzt zu einer Funktion $\sigma : \mathit{Term}(\Sigma, V) \rightarrow \mathit{Term}(\Sigma, V)$ durch $\sigma(f(t_1, \ldots, t_n) = f(\sigma(t_1, \ldots, \sigma(t_n))$

- Effektiv werden die Variablen im Term ersetzt.

Substitutionen werden auf Regeln und Gleichungen fortgesetzt:

- $\sigma(s \rightarrow t) = \sigma(s) \rightarrow \sigma(t)$
- $\sigma(s \simeq t) = \sigma(s) \simeq \sigma(t)$

Erinnerung: $\sigma(M) = \{\sigma(e) | e \in M\}$ für Mengen kennen wir schon!

Beispiel

Sei $\sigma_1 = \{x \leftarrow f(a), y \leftarrow a, z \leftarrow x\}$

Sei $\sigma_2 = \{x \leftarrow y, y \leftarrow z, z \leftarrow x\}$

$\sigma_1(f(g(x), f(y, g(z))))$	$\sigma_2(f(g(x), f(y, g(z))))$
$=f(\sigma_1(g(x)), \sigma_1(f(y, g(z))))$	$=f(\sigma_2(g(x)), \sigma_2(f(y, g(z))))$
$=f(g(\sigma_1(x)), f(\sigma_1(y), \sigma_1(g(z))))$	$=f(g(\sigma_2(x)), f(\sigma_2(y), \sigma_2(g(z))))$
$=f(g(f(a)), f(a, g(\sigma_1(z))))$	$=f(g(y), f(z, g(\sigma_2(z))))$
$=f(g(f(a)), f(a, g(x)))$	$=f(g(f(a)), f(a, g(x)))$

Instanz, Match

Definition

Seien $s, t \in Term(\Sigma, V)$ und seit σ eine Substitution.
Wenn $\sigma(s) = t$ gilt, so heißt

- σ ein Match von s auf t
- t eine Instanz von s

Sei nun t eine Instanz von s

- Wenn s keine Instanz von t ist, so heißt t eine echte Instanz von s

Instanz, Match

Beispiel

s	t	Match?	Echte Instanz?
x	$f(a,b)$	$\sigma = \{x \leftarrow f(a,b)\}$	Ja
x	$g(x)$	$\sigma = \{x \leftarrow g(x)\}$	Ja
$g(y)$	$g(x)$	$\sigma = \{y \leftarrow x\}$	Nein, $\{x \leftarrow y\}$
$f(x,g(x))$	$f(a,x)$	Nein	-
$f(x,g(y))$	$f(a,g(b))$	$\sigma = \{x \leftarrow a, y \leftarrow b\}$	Ja
$f(x,g(x))$	$f(a,h(a))$	Nein	-

Wenn ein Match existiert, so ist er (auf den beteiligten Variablen) eindeutig!

Matching

Frage: Wie findet man einen Match?

Idee: Traversiere s und t in Vorordnung, baue (Repräsentation von) Substitution σ

Situationen:

- s ist eine Variable
  - Fall 1: s ist in σ nicht gebunden: Erweitere σ um $s \leftarrow t$
  - Fall 2: s ist gebunden, $\sigma(s) = t$: Ok, matcht schon
  - Fall 3: s ist gebunden, $\sigma(s) \neq t$: Es gibt keinen Match!
- s ist von der Form $f(s_1, \ldots, s_n)$
  - Fall 1: t ist von der Form $f(t_1, \ldots, t_n)$: Mache mit den s_i, t_i weiter
  - Fall 2 : t ist von anderer Form: Es gibt keinen Match!

Matching

```
def matchTerms(s,t, sigma={}):
    if isVar(s):
        if not s in sigma:
            sigma[s] = t
            return sigma
        elif sigma[s] == t:
            return sigma
        else:
            return None
    elif isVar(t):
        return None
    elif getFun(s)!=getFun(t):
        return None
    else:
        for s,t in zip(getArgs(s), getArgs(t)):
            if matchTerms(s,t, sigma) == None:
                return None
        return sigma
```

Matching – Anmerkungen

Matching wird in vielen modenen Programmiersprachen eingesetzt:

- Haskell
- F#
- Python
- Aber auch: Finden der richtigen Methode bei Polymorphismus in C++

Matching wird in fast allen symbolischen Systemen eingesetzt:

- Expertensystem
- Planungssysteme
- Theorembeweiser
- Computeralgebrasystemen
- IDE's ("Intellisense")

Was ist ein Reduktionssystem?

Ein Reduktionssystem ist ein Tupel

$$(A, \rightarrow)$$

Hier: A ist Menge der Terme über einer Signatur

Was ist mit $\rightarrow$?

Rewrite- Relation

Definition

Seien Σ, V gegeben und sei R eine Menge von Regeln über $Term(\Sigma, V)$. R definiert the Rewrite- Relation $\rightarrow_R$ auf $Term(\Sigma, V)$ wie folgt:

$$s \rightarrow_R t$$

gdw

$$\exists p \in Pos(s), l{\rightarrow}r \in R, \sigma : s|_p = \sigma(l) \text{ und } t = s[\sigma(r)]_p$$

Wenn die linke Seite einer Regel auf einen Teilterm matched, kann dieser durch die instanziierte rechte Seite ersetzt werden

Programme als Terme (Beispiel)

Beispielprogramm

```
def fak1(a):
    if a==0:
        return 1
    else:
        return a*fak1(a-1)
```

Beispiel: Programme als Terme (Signatur Σ)

$arg\|_2$	Konstruktor für Argumentlisten
$stmt\|_2$	Konstruktor für Befehlslisten
$end\|_0$	Listenende
$fcall\|_2$	Funktionsaufruf
$def\|_3$	Kodiert **def**
$return\|_1$	Kodiert **return**
$equal\|_2$	Kodiert **==**
$mult\|_2$	Kodiert *****
$minus\|_2$	Kodiert **-**
$fak1\|_0$	Name aus dem Programm
$a\|_0$	Name aus dem Programm
$0\|_0$	Konstante aus dem Programm
$1\|_0$	Konstante aus dem Programm

Programme als Terme (Kodiertes Program)

```
def(fak1, arg(a, end),
    stmt(if(equal(a, 0),
            stmt(return(1),
                 end),
            stmt(return(mult(a,
                             fcall(fak1,
                                   arg(minus(a,1)),
                                       end)),
                 end),
        end)
```

Programme als Terme (Rewriting)

Mögliche Anwendung: Optimierung

- $mult(x, 0) \rightarrow 0$
- $mult(0, x) \rightarrow 0$
- $mult(x, 1) \rightarrow x$
- $mult(1, x) \rightarrow x$
- $minus(x, 0) \rightarrow x$

… aber auch denkbar:

- Regeln, die nutzlose **else**- Statements entfernen
- Regeln, die konstante Ausdrücke auswerten
- Regeln, die tote **if**/**else** Zweige entfernen

Zusammenfassung

Beispiel: Rewrite-System

Terme

- Positionen
- Grundterme
- Substitutionen

Matching

- Instanzen
- Matching-Algorithmus

Definition Rewrite- Relation $\rightarrow_R$

Anwendungsbeispiel: Programm als Term

Aufgaben

- Zeigen Sie Lemma (Pos- 1 (b)): Sei $t \in Term(\Sigma, V)$ und $pq \in Pos(t)$. Dann gilt: $(t|_p)|_q = t|_{pq}$.
- Welche der folgenden Term sind Instanzen/echte Instanzen voneinander? Geben Sie jeweils den Match an.
  $t_1 = h(f(x, a), g(x))$, $t_2 = h(x, f(y, g(x)))$, $t_3 = x$,
  $t_4 = h(f(y, z), g(z))$
- Betrachten Sie
  $R = \{f(f(x, y), z) \rightarrow f(x, f(y, z)), f(x, i(x)) \rightarrow e, f(x, e) \rightarrow e\}$.
  Bestimmen Sie alle $\rightarrow_R$ Normalformen der folgenden Terme:
  - $f(f(a, i(a)), z)$
  - $f(f(x, i(y)), e)$
  - $f(f(e, i(e)), i(e))$