Entwicklung eines Termersetzungssystems für assoziative und kommutative Ausdrücke

Vortrag zur Bachelorarbeit

Bruno Borchardt

18. August 2021

Betreuer: PD Dr. Prashant Batra
Prof. Dr. Siegfried Rump

Motivation

- Ziel: vereinfache mathematische Ausdrücke automatisch, etwa $\sin(\frac{21}{2}\pi)$ zu (exakt) 1 oder $x^2 + 5xy - 12xy$ zu $x^2 - 7xy$

Motivation

- Ziel: vereinfache mathematische Ausdrücke automatisch, etwa $\sin(\frac{21}{2}\pi)$ zu (exakt) 1 oder $x^2 + 5xy - 12xy$ zu $x^2 - 7xy$
- Problem: Computer kennen keine mathematischen Gesetzmäßigkeiten

Motivation

- Ziel: vereinfache mathematische Ausdrücke automatisch, etwa $\sin(\frac{21}{2}\pi)$ zu (exakt) 1 oder $x^2 + 5xy - 12xy$ zu $x^2 - 7xy$
- Problem: Computer kennen keine mathematischen Gesetzmäßigkeiten
- Idee: formuliere Vereinfachungsregeln

Motivation

- Ziel: vereinfache mathematische Ausdrücke automatisch, etwa $\sin(\frac{21}{2}\pi)$ zu (exakt) 1 oder $x^2 + 5xy - 12xy$ zu $x^2 - 7xy$
- Problem: Computer kennen keine mathematischen Gesetzmäßigkeiten
- Idee: formuliere Vereinfachungsregeln
- Mathematica ist Beispiel einer kommerziellen Umsetzung

Term

Definition

Terme sind

- *Konstantensymbole*, etwa `1`, `-3i`, `a`, `sin`
- oder *Funktionsanwendungen* eines Terms f auf die Terme $t_1, \ldots, t_n$, geschrieben $f(t_1, \ldots, t_n)$, etwa `sin(3)`

Term

Definition

Terme sind

- *Konstantensymbole*, etwa `1`, `-3i`, `a`, `sin`
- oder *Funktionsanwendungen* eines Terms f auf die Terme $t_1, \ldots, t_n$, geschrieben $f(t_1, \ldots, t_n)$, etwa `sin(3)`

Ist $t = 3 + 4a$ ein Term?

Term

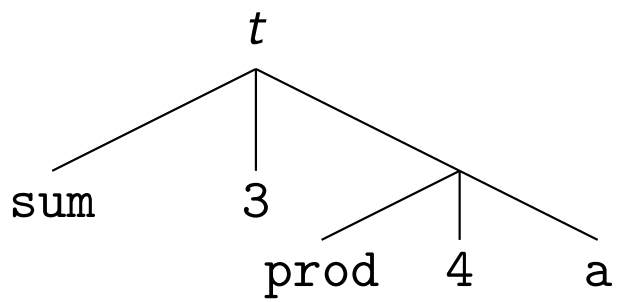
Definition

Terme sind

- *Konstantensymbole*, etwa `1`, `-3i`, `a`, `sin`
- oder *Funktionsanwendungen* eines Terms f auf die Terme $t_1, \ldots, t_n$, geschrieben $f(t_1, \ldots, t_n)$, etwa `sin(3)`

Ist $t = 3 + 4a$ ein Term?

Ja, formal geschrieben
$t =$ `sum(3, prod(4, a))`

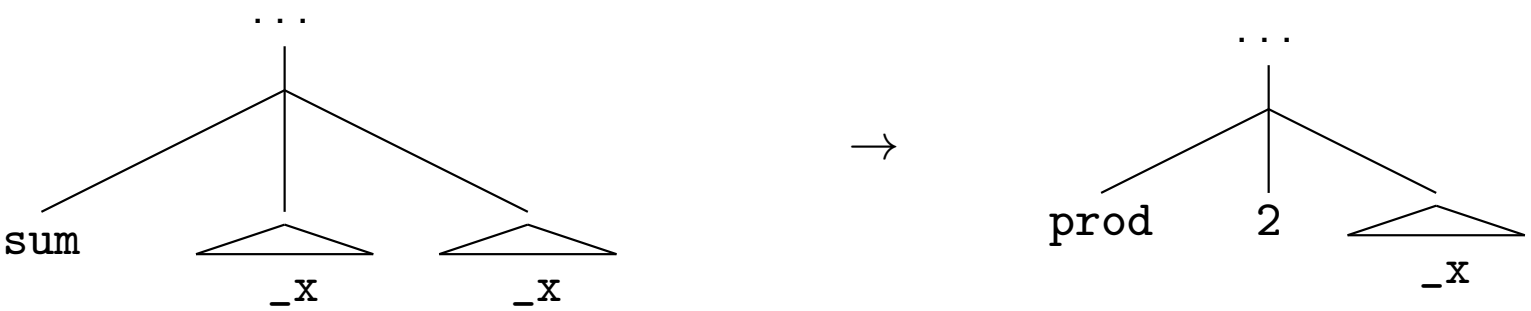


Ersetzungsregeln

- Paar (l, r) aus Termen l und r, geschrieben $l = r$
- unterscheide *Literale* (zu vereinfachende Terme) von *Mustern* (Terme in Regeln)
- wird l in einem Literal gefunden, erfolgt die Ersetzung durch r
- *Mustervariablen* sind Platzhalter in Regeln, idetifiziert durch vorangestellten Unterstrich _, etwa `_x`

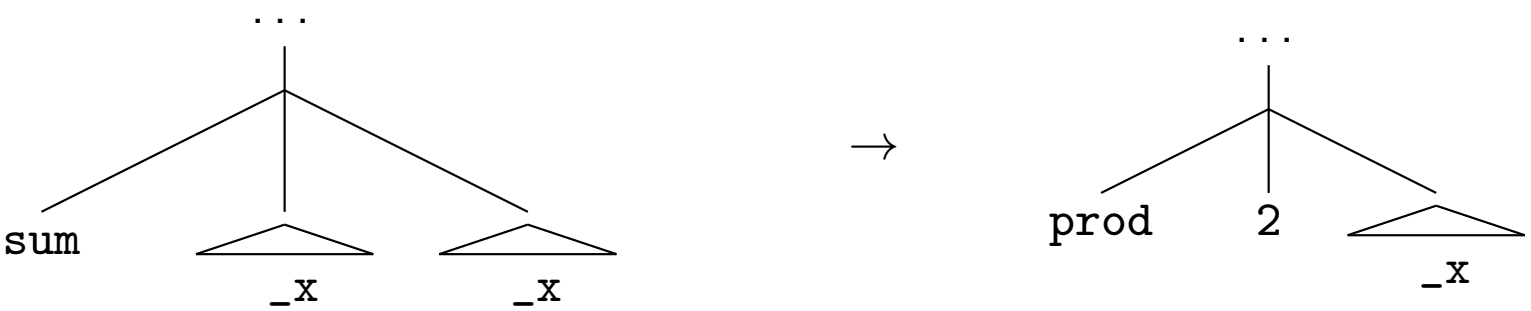
Beispiel

Die Regel `sum(_x, _x) = prod(2, _x)` steht für die folgende Transformation eines Literals.

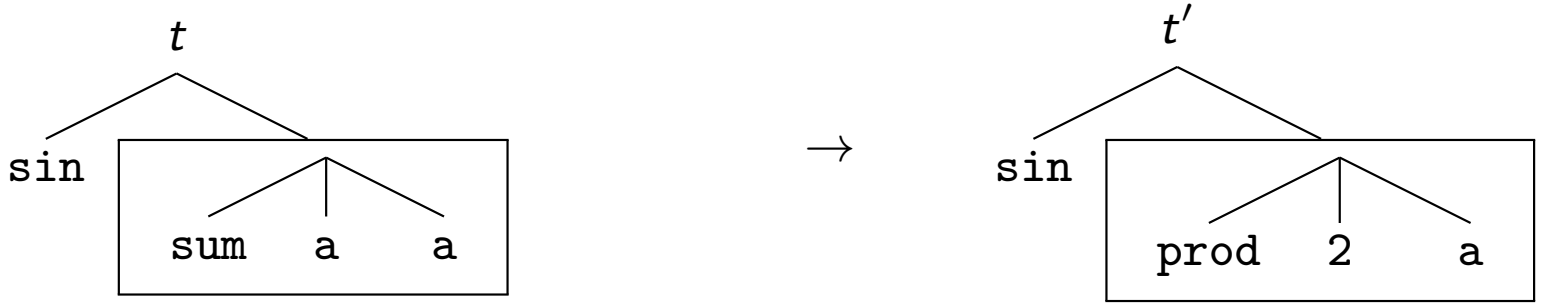


Beispiel

Die Regel `sum(_x, _x) = prod(2, _x)` steht für die folgende Transformation eines Literals.



Aus $t =$ `sin(sum(a, a))` wird $t' =$ `sin(prod(2, a))`.



Normalisierung

Wann sind zwei Terme identisch?

1. `sum(1, 1)` vs. `2`
2. `sum(a, b)` vs. `sum(b, a)`
3. `sum(sum(a, b), 2)` vs. `sum(a, sum(b, 2))`

Normalisierung

Wann sind zwei Terme identisch?

1. `sum(1, 1)` vs. `2`
2. `sum(a, b)` vs. `sum(b, a)`
3. `sum(sum(a, b), 2)` vs. `sum(a, sum(b, 2))`

1. werte bekannte Funktionsanwendungen aus:
   `sum(1, 1)` $\rightarrow$ `2`
2. sortiere Argumente kommutativer Funktionsanwendungen:
   `sum(b, a)` $\rightarrow$ `sum(a, b)`
3. erlaube keine geschachtelten Funktionsanwendungen derselben assoziativen Funktion:
   `sum(sum(a, b), 2)` $\rightarrow$ `sum(a, b, 2)`

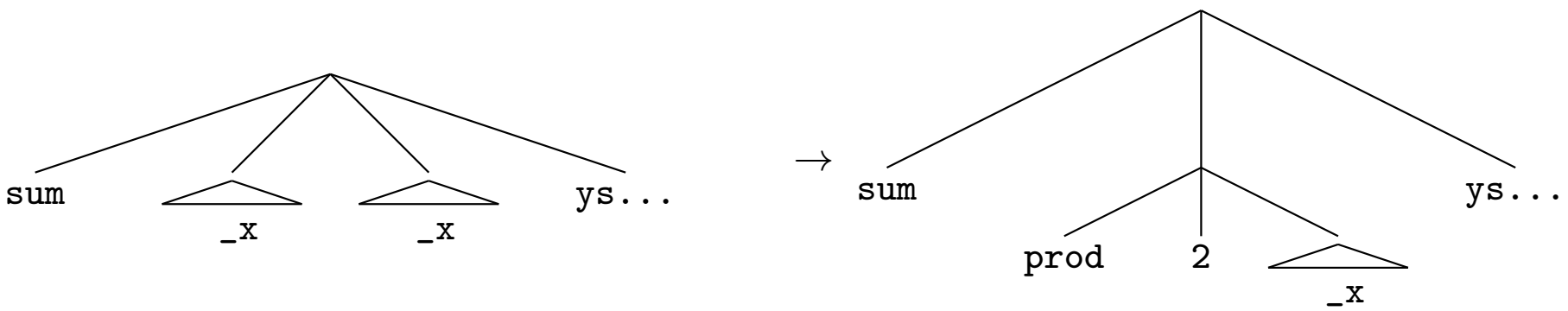
Multi-Mustervariablen

- Bestimmte Muster sollen Funktionsanwendungen beliebiger Argumentanzahl darstellen können.
- Multi-Mustervariablen sind Platzhalter für beliebig viele Argumente, identifiziert durch einen Suffix aus drei Punkten, etwa `xs...`

Multi-Mustervariablen

- Bestimmte Muster sollen Funktionsanwendungen beliebiger Argumentanzahl darstellen können.
- Multi-Mustervariablen sind Platzhalter für beliebig viele Argumente, identifiziert durch einen Suffix aus drei Punkten, etwa `xs...`

Regel `sum(_x, _x) = prod(2, _x)` wird zu
`sum(_x, _x, ys...) = sum(prod(2, _x), ys...)`.



Termersetzungssystem

Pseudocode Termersetzungssystem

Input: Regelmenge R, Literal t

1. normalisiere t
2. Wenn $\exists(l, r) \in R$ anwendbar auf t oder Teil von t
3. ersetze l in t durch r
4. gehe zu 1
5. gebe t zurück

Demo

Demo

Spickzettel

```
diff(3 + t + a, t)

:add diff(_u + vs..., _x) = diff(_u, _x) + diff(sum(vs...), _x)

:add diff(_y, _x) | !contains(_y, _x) = 0

:add diff(_x, _x) = 1
```

Match

Definition

Eine Funktion v, die Mustervariablen auf Literale abbildet, ist ein *Match* des Musters p mit dem Literal t, wenn die normalisierte Form der Ersetzung der Mustervariablen x_i in p durch ihre Funktionswerte $v(x_i)$ mit t identisch ist.

Match

Definition

Eine Funktion v, die Mustervariablen auf Literale abbildet, ist ein *Match* des Musters p mit dem Literal t, wenn die normalisierte Form der Ersetzung der Mustervariablen x_i in p durch ihre Funktionswerte $v(x_i)$ mit t identisch ist.

Beispiel 1

Mit $v(\texttt{\_x}) = \texttt{sin(a)}$ ist v ein Match des Musters $p = \texttt{sum(\_x, \_x)}$ mit dem Literal $t = \texttt{sum(sin(a), sin(a))}$.

Match

Definition

Eine Funktion v, die Mustervariablen auf Literale abbildet, ist ein *Match* des Musters p mit dem Literal t, wenn die normalisierte Form der Ersetzung der Mustervariablen x_i in p durch ihre Funktionswerte $v(x_i)$ mit t identisch ist.

Beispiel 1

Mit $v(\texttt{\_x}) = \texttt{sin(a)}$ ist v ein Match des Musters $p = \texttt{sum(\_x, \_x)}$ mit dem Literal $t = \texttt{sum(sin(a), sin(a))}$.

Beispiel 2

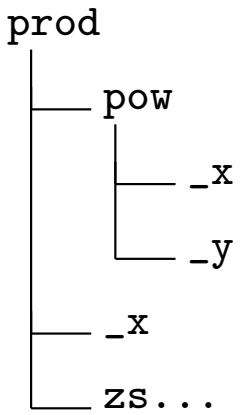
Mit $v(\texttt{\_x}) = 3$ ist v ein Match des Musters $p = \texttt{sum(\_x, \_x)}$ mit dem Literal $t = 6$.

Vorgehen

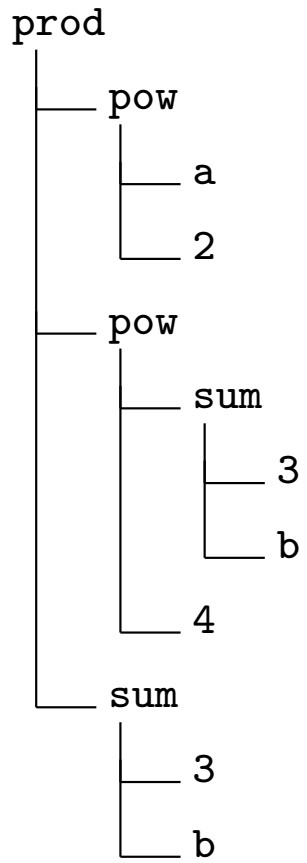
- Muster und Literal werden parallel abgelaufen
- Mustervariablen werden in Entdeckungsreihenfolge der Tiefensuche an Teilterme des Literals gebunden
- Backtracking wenn Match aktuell unmöglich
- Algorithmus berücksichtigt Kommutativität entsprechender Funktionsanwendungen

Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`

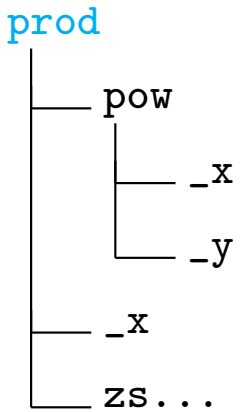


Matchstatus	
_x	-
_y	-
zs...	-

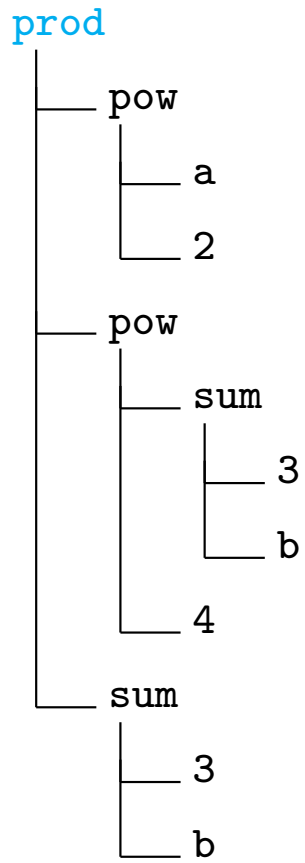


Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`

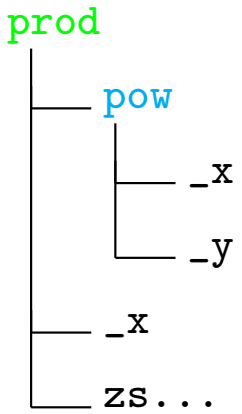


Matchstatus	
_x	-
_y	-
zs...	-

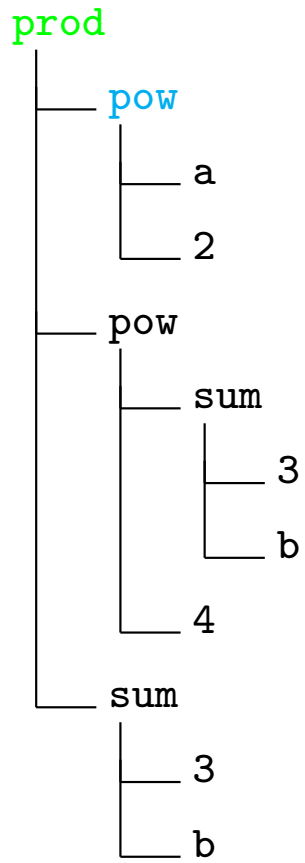


Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`

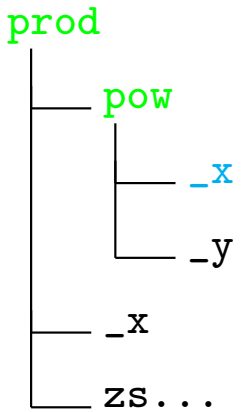


Matchstatus	
_x	-
_y	-
zs...	-



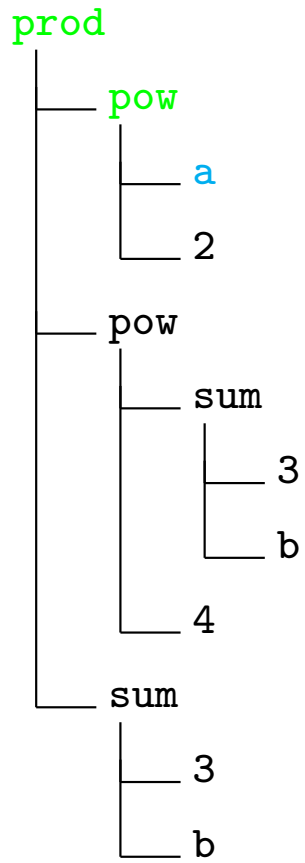
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



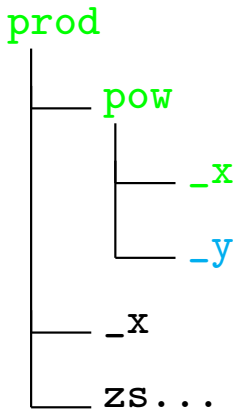
Matchstatus

_x	a
_y	-
zs...	-



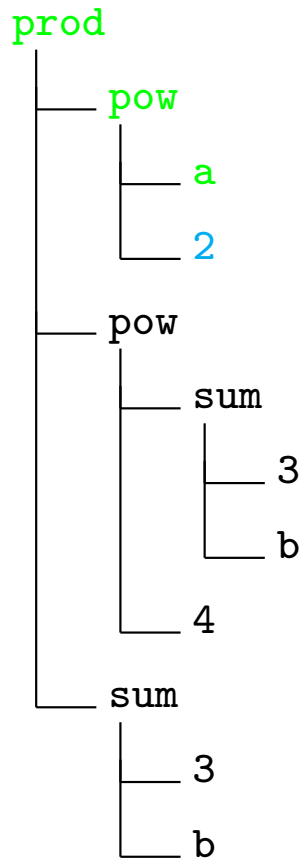
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



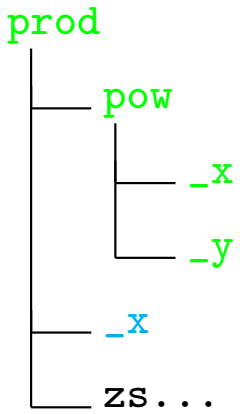
Matchstatus

_x	a
_y	2
zs...	-



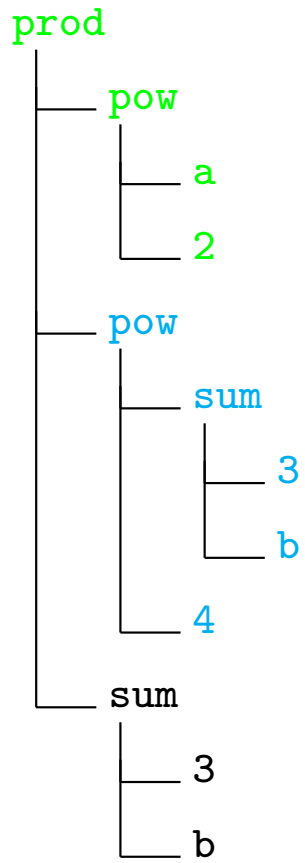
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



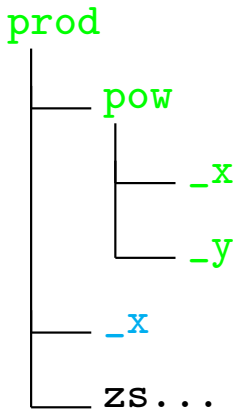
Matchstatus

_x	a
_y	2
zs...	-

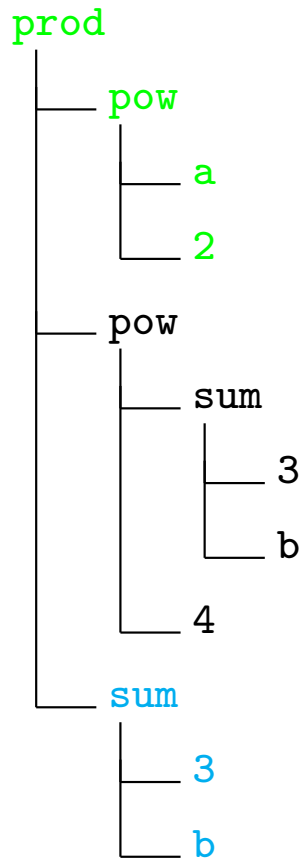


Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`

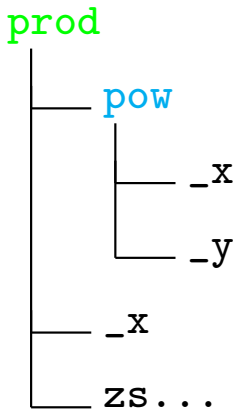


Matchstatus	
_x	a
_y	2
zs...	-

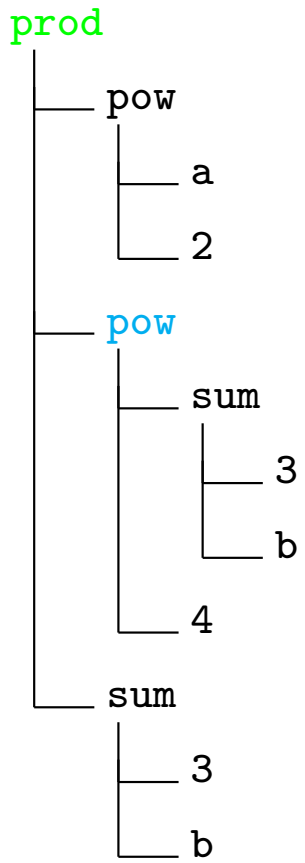


Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`

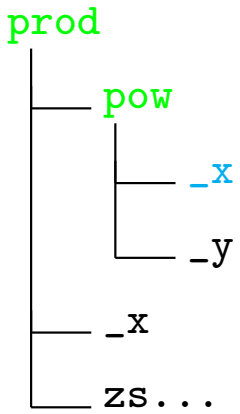


Matchstatus	
_x	-
_y	-
zs...	-



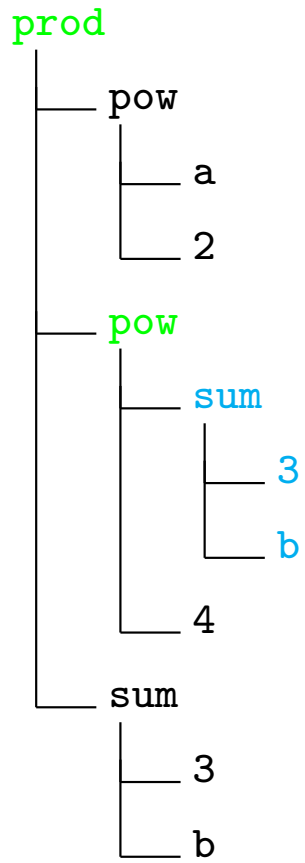
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



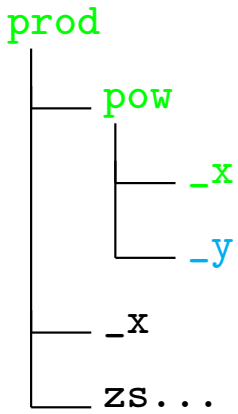
Matchstatus

_x	sum(3, b)
_y	-
zs...	-



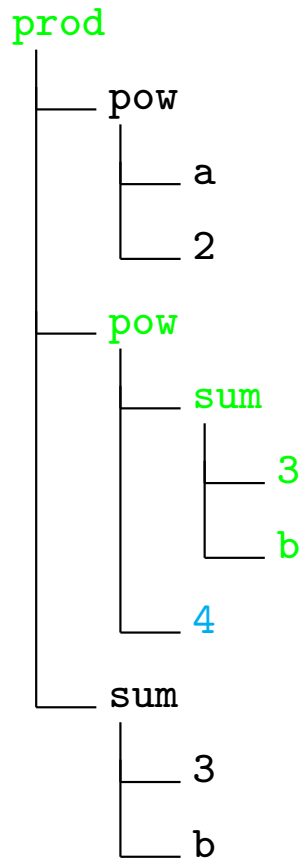
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



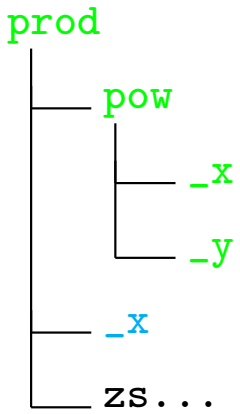
Matchstatus

_x	sum(3, b)
_y	4
zs...	-



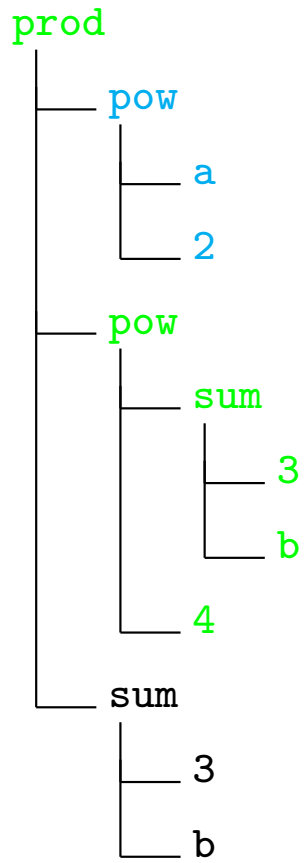
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



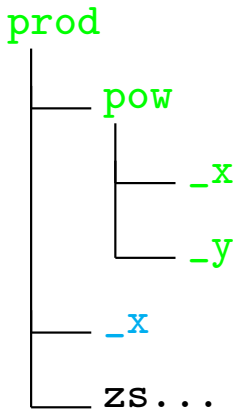
Matchstatus

_x	sum(3, b)
_y	4
zs...	-



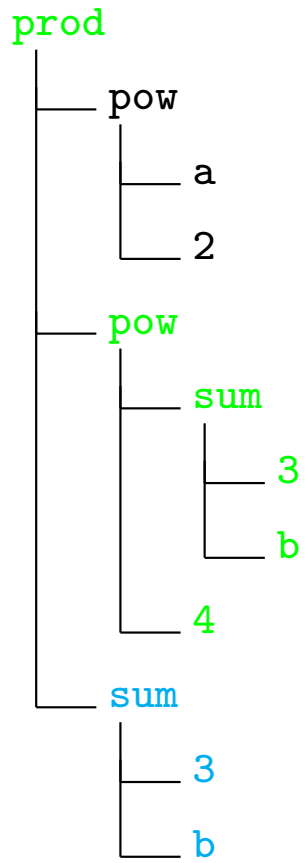
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



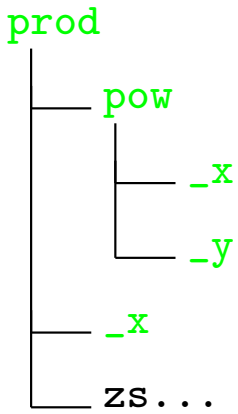
Matchstatus

_x	sum(3, b)
_y	4
zs...	-



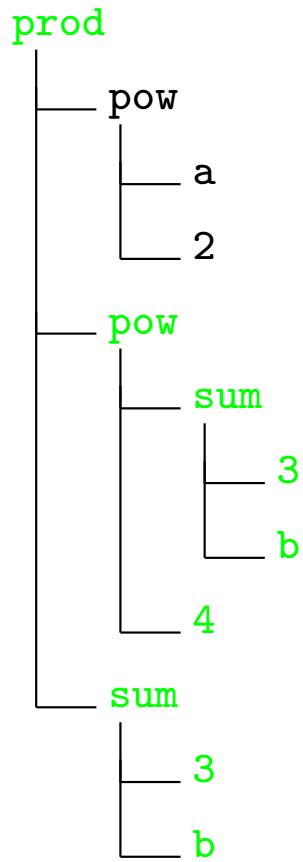
Beispiel

Matche $p =$ `prod(pow(_x, _y), _x, zs...)` in
$t =$ `prod(pow(a, 2), pow(sum(3, b), 4), sum(3, b))`



Matchstatus

_x	sum(3, b)
_y	4
zs...	pow(a, 2)



Demo

Zusammenfassung

- Terme als Baumstrukturen
- Ersetzungsregeln als Paare von Mustertermen
- Mustererkennung für assoziative und kommutative Funktionen teilweise rechenintensiv
  - Assoziativität indirekt über Multi-Mustervariablen ausgedrückt
  - Kommutativität direkt berücksichtigt
  - Backtracking

Fragen?