# Komposition von Baumübersetzern als Programmtransformation

**Janis Voigtländer**

# Überblick

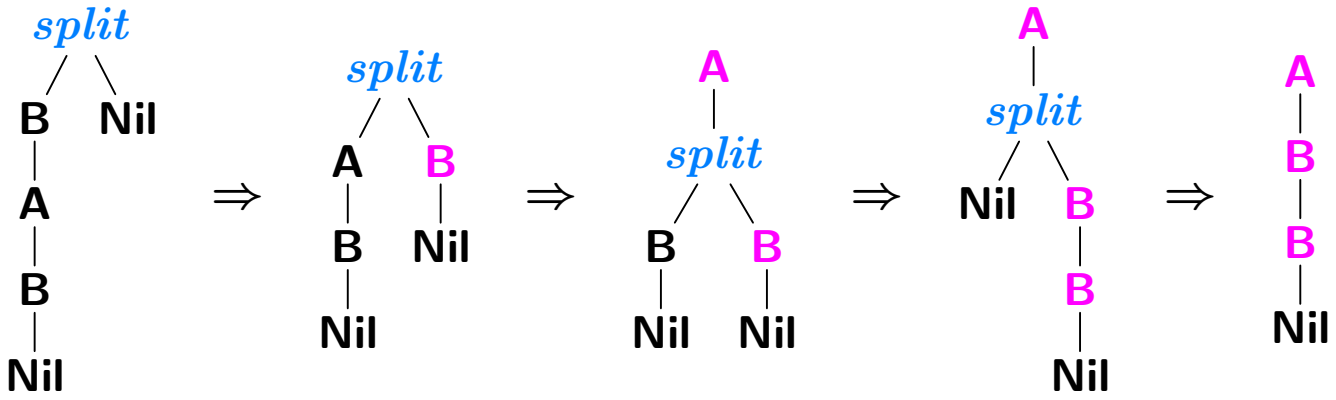
1. **Funktionale Programme**
2. **Zwischenergebnisse und ihre Elimination**
3. **Formale Effizienzanalyse**
4. **Verwandte Arbeiten**

# Funktionale Programme

- **deklarative Spezifikationen, aber ausführbar**
- **keine Seiteneffekte $\Rightarrow$ referentielle Transparenz**
- **kompositionelle Semantik $\Rightarrow$ gleichungsbasiertes Schließen**
- **hohes Potential zur Modularisierung von Programmen**

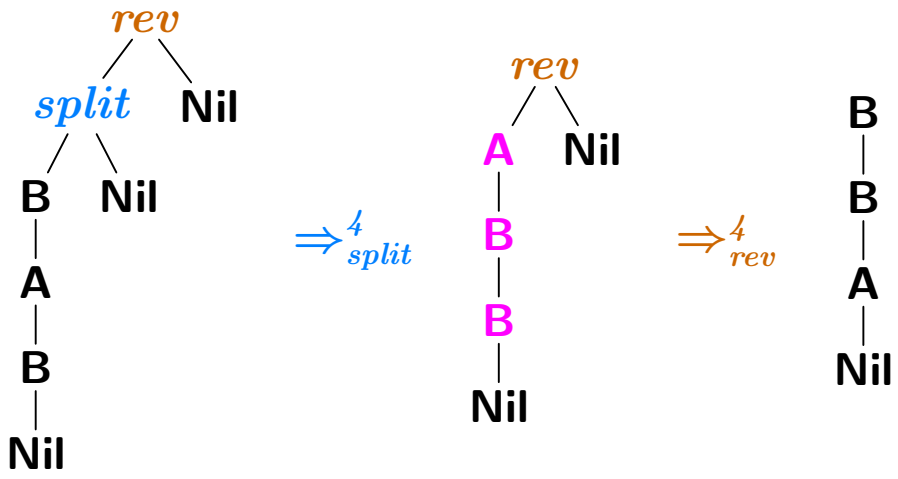
# Strukturelle Rekursion und akkumulierende Parameter

```
data List = A List | B List | Nil
split :: List → List → List
split (A u) y = A (split u y)
split (B u) y = split u (B y)
split  Nil  y = y
```



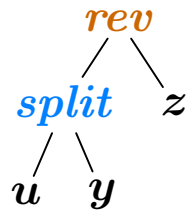
# Modularität vs. Effizienz

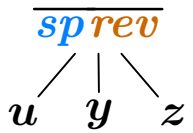
$$
\begin{aligned}
&\mathit{rev} :: \mathsf{List} \to \mathsf{List} \to \mathsf{List}\\
&\mathit{rev}\ (\mathsf{A}\ v)\ z = \mathit{rev}\ v\ (\mathsf{A}\ z)\\
&\mathit{rev}\ (\mathsf{B}\ v)\ z = \mathit{rev}\ v\ (\mathsf{B}\ z)\\
&\mathit{rev}\ \ \mathsf{Nil}\ \ z = z\\
&\mathit{main}\ t = \mathit{rev}\ (\mathit{split}\ t\ \mathsf{Nil})\ \mathsf{Nil}
\end{aligned}
$$



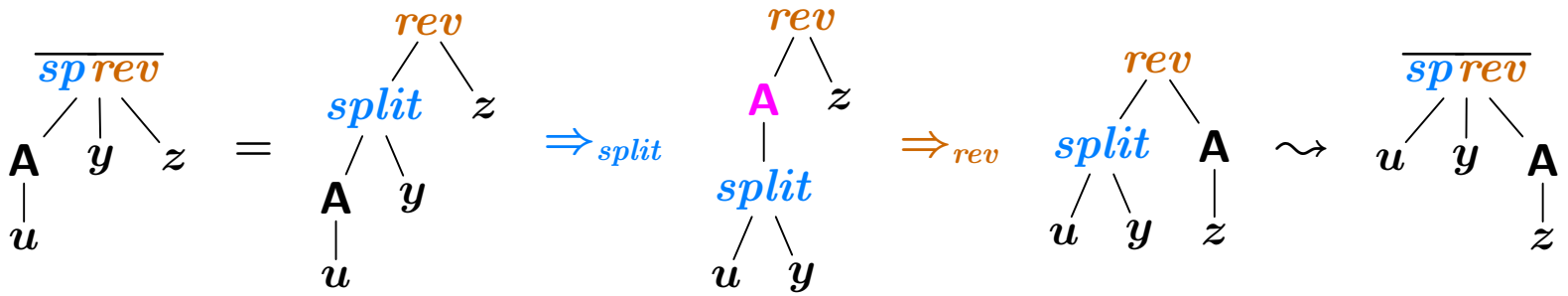

**Zwischenergebnisse führen zu Ineffizienzen!**

# Klassische Deforestation [Wadler, 1990]

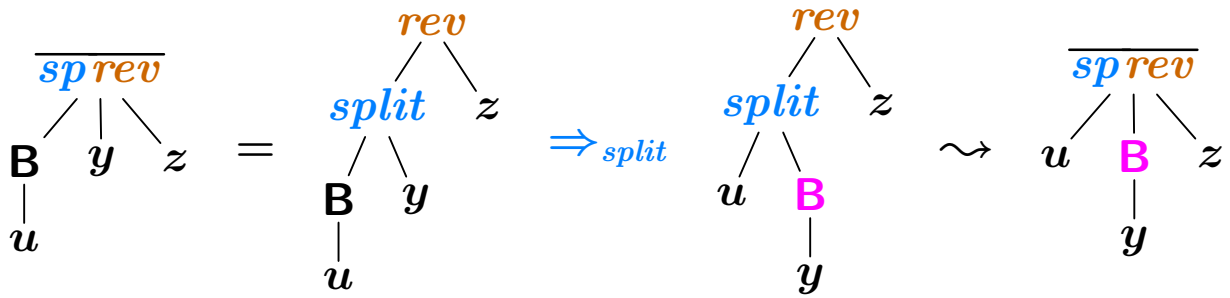
**Hauptideen: „Folding“ von** 

**zu** 

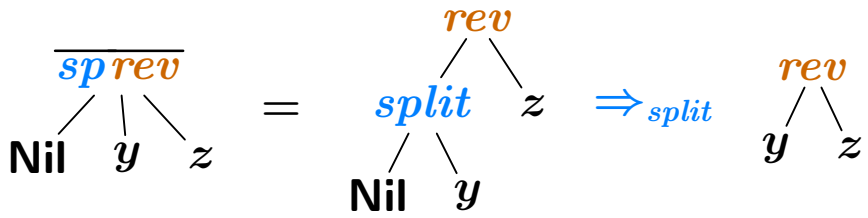
**und „Übersetzung“ rechter Seiten von *split* mit Regeln von *rev*:**

**1.** 

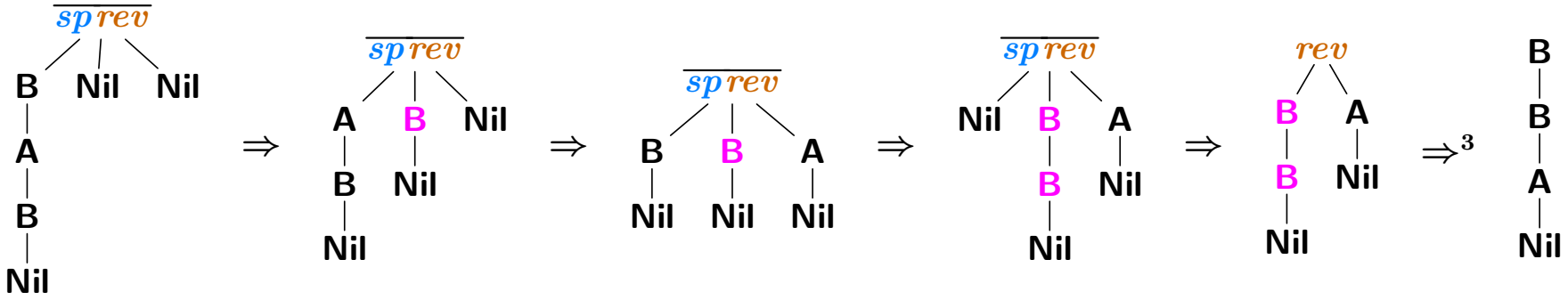
**2.**




**3.**

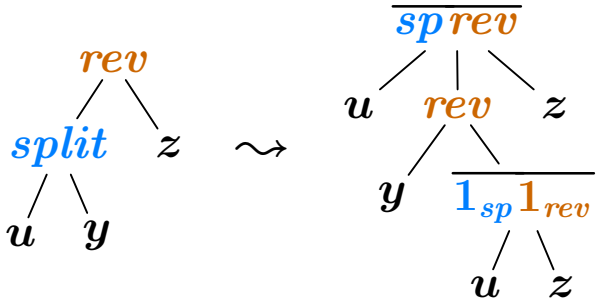



**Nur ein Teil des Zwischenergebnisses entfernt:**



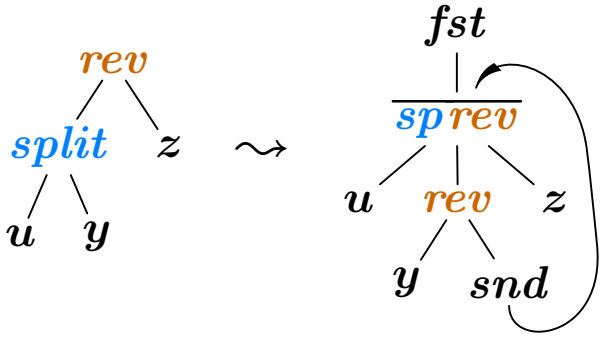
# Deforestation in akkumulierenden Parametern: zwei Lösungen

1. **Verwendung von Hilfsfunktionen:**




[V. & Kühnemann, 2004] Composition of functions with accumulating parameters. *Journal of Functional Programming*, 14:317–363.

2. **Verwendung von Tupling und zirkulären Bindungen:**




[V., 2004] Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17:129–163. (Auch: *Proc. ASIA-PEPM'02*, pp. 126–137, ACM Press.)

# Transformiertes Programm (nach 1.)

$$
\begin{array}{l}
\overline{sprev}, \overline{1_{sp}1_{rev}} :: \mathsf{List} \to \mathsf{List} \to \mathsf{List} \\
\overline{sprev}\ (\mathsf{A}\ u)\ y' = \overline{sprev}\ u\ y' \\
\overline{sprev}\ (\mathsf{B}\ u)\ y' = \overline{sprev}\ u\ y' \\
\overline{sprev}\ \ \mathsf{Nil}\ \ y' = y' \\
\overline{1_{sp}1_{rev}}\ (\mathsf{A}\ u)\ z = \overline{1_{sp}1_{rev}}\ u\ (\mathsf{A}\ z) \\
\overline{1_{sp}1_{rev}}\ (\mathsf{B}\ u)\ z = \mathsf{B}\ (\overline{1_{sp}1_{rev}}\ u\ z) \\
\overline{1_{sp}1_{rev}}\ \ \mathsf{Nil}\ \ z = z \\
\\
main'\ t = \overline{sprev}\ t\ (\overline{1_{sp}1_{rev}}\ t\ \mathsf{Nil})
\end{array}
$$

**Aber was gilt bezüglich der Effizienz?**

# Formale Effizienzanalyse

- **Maß: Anzahl der call-by-need Reduktionsschritte**
- **Ansatz:**
  - **annotiere Original- und komponiertes Programm, um Reduktionsschritte in der Ausgabe widerzuspiegeln**
  - **führe Annotation des komponierten Programms „rückwärts“ durch die Kompositionskonstruktion**
  - **vergleiche (und manipuliere) Annotationen des Originalprogramms, um hinreichende Bedingungen herzuleiten**

**[V., 2002b] Conditions for efficiency improvement by tree transducer composition. *Proc. RTA'02*, LNCS 2378:222–236. Springer-Verlag.**

# Der Schlüssel zur Effizienzanalyse...

**Anreichern der Regeln der Kompositionspartner $M_1$ und $M_2$ zu:**

---

$R_1^{\rightarrow \diamond\star}$:

$$f\ (\sigma\ u_1 \cdots u_p)\ y_1 \cdots y_r = \diamond\ rhs_{M_1,f,\sigma}[y_k \leftarrow \star\ y_k \mid 1 \leq k \leq r]$$

---

$R_2^{\diamond\star \rightarrow \circ}$:

$$g\ (\delta\ v_1 \cdots v_q)\quad z_1 \cdots z_s = rhs_{M_2,g,\delta}$$

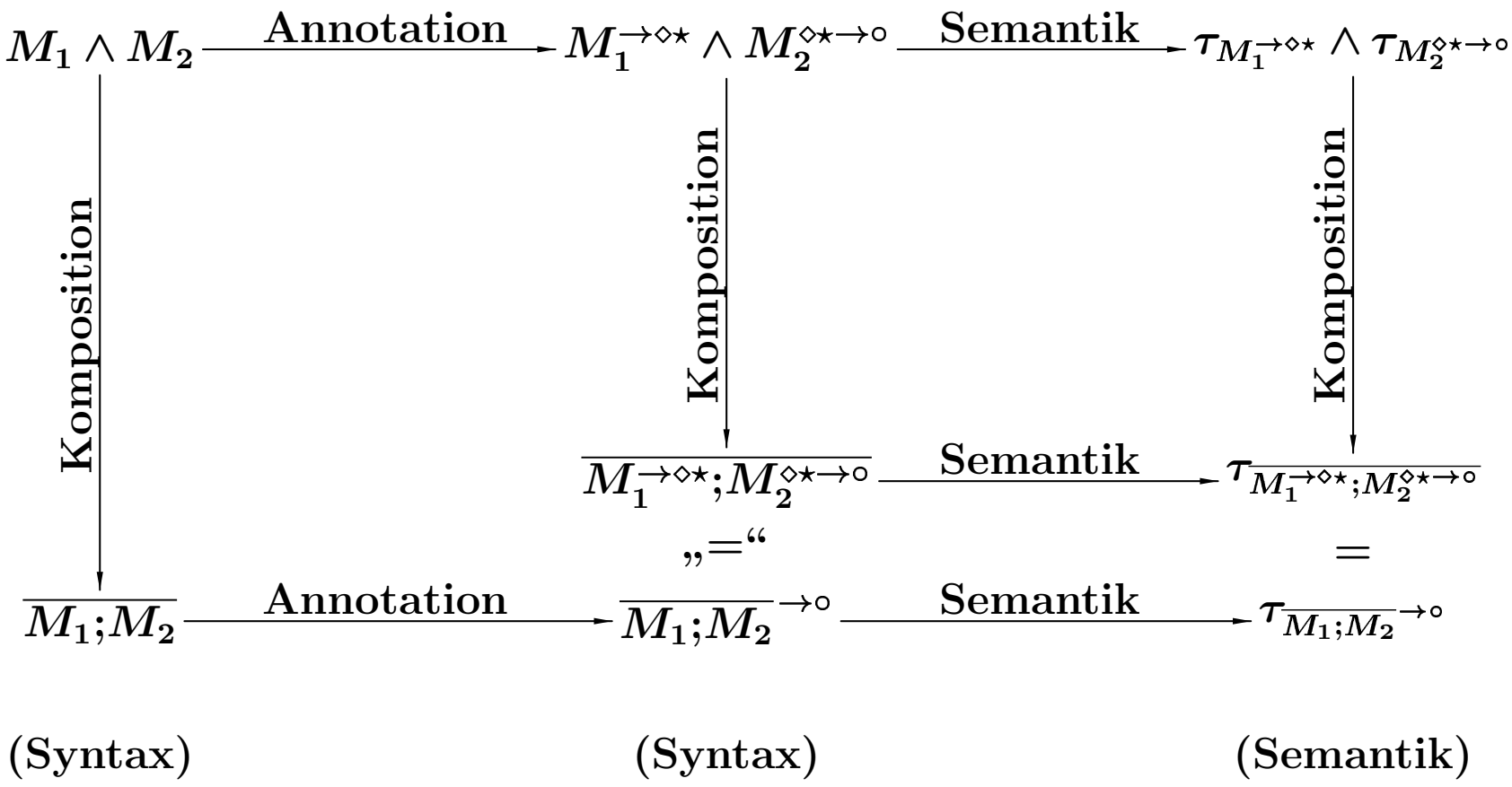
$$g\ (\diamond\ v_1)\qquad z_1 \cdots z_s = \circ\ (g\ v_1\ z_1 \cdots z_s)$$

$$g\ (\star\ v_1)\qquad z_1 \cdots z_s = g\ v_1\ (\circ\ z_1)\ \cdots\ (\circ\ z_s)$$

---

**erlaubt die Berechnung der Anzahl der vom *komponierten Programm* für eine bestimmte Eingabe durchgeführten *call-by-name* Schritte mittels $\tau_{M_1}{}^{\rightarrow\diamond\star}$ ; $\tau_{M_2}{}^{\diamond\star\rightarrow\circ}$ ; $|\,.\,|_\circ$.**

# . . . und dessen Beweis

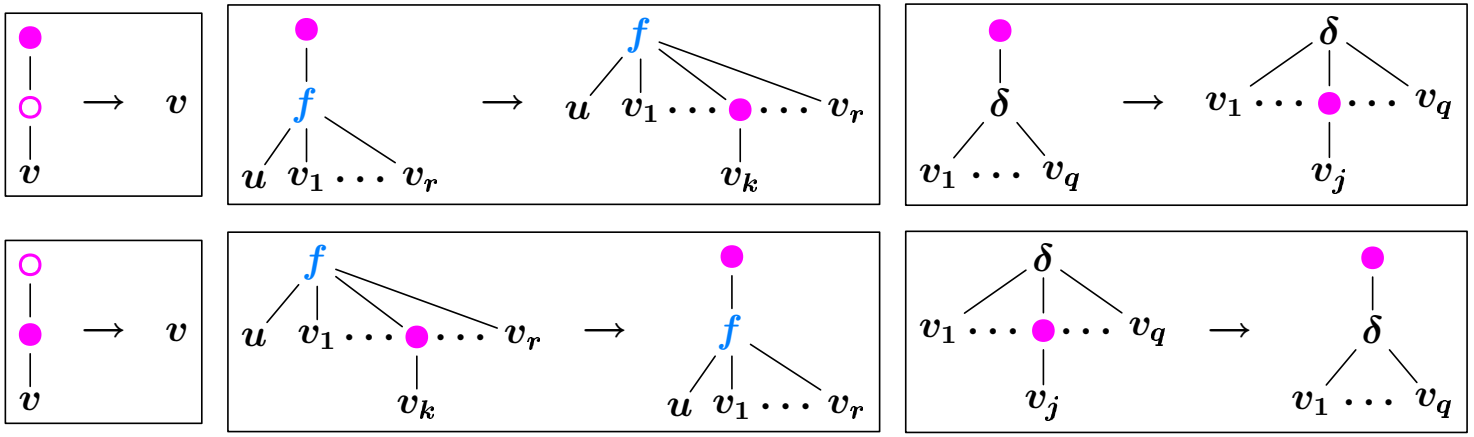


# Schließlich, am Beispiel:

**Annotiertes Programm:**

$$
\begin{array}{lll}
\mathit{split}\ (\mathsf{A}\ u)\ y = \bullet\ (\mathsf{A}\ (\mathit{split}\ u\ (\circ\ y))) & & \mathit{rev}\ (\mathsf{A}\ v)\ z = \mathit{rev}\ v\ (\mathsf{A}\ z) \\
\mathit{split}\ (\mathsf{B}\ u)\ y = \mathit{split}\ u\ (\circ\ (\bullet\ (\mathsf{B}\ y))) & & \mathit{rev}\ (\mathsf{B}\ v)\ z = \mathit{rev}\ v\ (\mathsf{B}\ z) \\
\mathit{split}\ \ \mathsf{Nil}\ \ y = y & & \mathit{rev}\ \ \mathsf{Nil}\ \ z = z \\
 & & \mathit{rev}\ (\circ\ v)\ z = \circ\ (\mathit{rev}\ v\ z) \\
\mathit{main}\ t = \mathit{rev}\ (\mathit{split}\ t\ (\circ\ (\bullet\ \mathsf{Nil})))\ \mathsf{Nil} & & \mathit{rev}\ (\bullet\ v)\ z = \bullet\ (\mathit{rev}\ v\ z)
\end{array}
$$

**Da *split* *context-linear* und *-nondeleting*, sowie *rev* *linear* und *nondeleting* ist, können alle folgenden Regeln mit dem Ziel benutzt werden, alle ∘-Symbole in den rechten Seiten von *split* zu eliminieren:**



## Weiterhin von praktischer Bedeutung:

- **Effiziente Prozedur zur Entscheidung der Effizienzkriterien**
- **Betrachtung der Effizienz unter partieller Berechnung und (ansatzweise) unter Berücksichtigung von Tail-Calls**
- **Korrektheit der Transformation auf partiell definierten und auf unendlichen Bäumen**

# Verwandte Arbeiten

- **Elimination von Zwischenergebnissen:**
  - **klassische Deforestation [Wadler, 1990], [Chin, 1994]**
  - **Shortcut-Def./Freie Theoreme [Gill *et al.*, 1993], [Svenningsson, 2002], [V., 2002a], [Johann & V., 2004]**
  - **Tree Transducer Composition [Engelfriet & Vogler, 1985], [Kühnemann, 1998 & 1999], [Maneth, 2003]**
  - **via Attributgrammatiken [Correnson *et al.*, 1999]**
  - **zirkuläre Programme [V., 2002c & 2004], [Nishimura, 2002 & 2004]**
- **Formale Effizienzanalyse für Transformationstechniken:**
  - **für klassische Deforestation [Sands, 1996]**
  - **für Tree Transducer Composition [Kühnemann, 1999], [Höff, 1999]**
- **Programmtransformation basierend auf Theorie der Tree Transducer:**
  - **Substitutionselimination [Kühnemann *et al.*, 2001], [Perst & Seidl, 2004]**
  - **Elimination akkumulierender Parameter [Giesl, Kühnemann & V., 2003]**

## Begutachtete Veröffentlichungen

[Voigtländer, 2002a] Concatenate, reverse and map vanish for free. *7th International Conference on Functional Programming, Proceedings*, SIGPLAN Notices 37(9):14–25. ACM Press.

[Voigtländer, 2002b] Conditions for efficiency improvement by tree transducer composition. *13th International Conference on Rewriting Techniques and Applications, Proceedings*, LNCS 2378:222–236. Springer-Verlag.

[Voigtländer, 2002c] Using circular programs to deforest in accumulating parameters. *Asian Symposium on Partial Evaluation and Semantics-Based Program Manipulation, Proceedings*, pages 126–137. ACM Press.

[Giesl, Kühnemann & Voigtländer, 2003] Deaccumulation — Improving provability. *8th Asian Computing Science Conference, Proceedings*, LNCS 2896:146–160. Springer-Verlag.

[Johann & Voigtländer, 2004] Free theorems in the presence of *seq*. *31st Symposium on Principles of Programming Languages, Proceedings*, SIGPLAN Notices 39(1):99–110. ACM Press.

---

[Voigtländer & Kühnemann, 2004] Composition of functions with accumulating parameters. *Journal of Functional Programming*, 14(3):317–363. Cambridge University Press.

[Voigtländer, 2004] Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17(1–2):129–163. Kluwer Academic Publishers.

# Literatur


[Adámek, 2003] On final coalgebras of continuous functors. *Theoretical Computer Science*, 294(1–2):3–29.

[Barr, 1993] Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114(2):299–315.

[Chin, 1994] Safe fusion of functional expressions II: Further improvements. *Journal of Functional Programming*, 4(4):515–555.

[Correnson, Duris, Parigot & Roussel, 1999] Declarative program transformation: A deforestation case-study. *Principles and Practice of Declarative Programming, Proceedings*, LNCS 1702:360–377. Springer-Verlag.

[Engelfriet & Vogler, 1985] Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146.

[Gill, Launchbury & Peyton Jones, 1993] A short cut to deforestation. *Functional Programming Languages and Computer Architecture, Proceedings*, pages 223–232. ACM Press.

[Höff, 1999] *Vergleich von Verfahren zur Elimination von Zwischenergebnissen bei funktionalen Programmen.* Master's thesis, Technische Universität Dresden.

[Kühnemann, 1998] Benefits of tree transducers for optimizing functional programs. *Foundations of Software Technology and Theoretical Computer Science, Proceedings*, LNCS 1530:146–157. Springer-Verlag.

[Kühnemann, 1999] Comparison of deforestation techniques for functional programs and for tree transducers. *Functional and Logic Programming, Proceedings*, LNCS 1722:114–130. Springer-Verlag.

[Kühnemann, Glück & Kakehi, 2001] Relating accumulative and non-accumulative functional programs. *Rewriting Techniques and Applications, Proceedings*, LNCS 2051:154–168. Springer-Verlag.

[Maneth, 2003] The macro tree transducer hierarchy collapses for functions of linear size increase. *Foundations of Software Technology and Theoretical Computer Science, Proceedings*, LNCS 2914:326–337. Springer-Verlag.

[Nishimura, 2002] Deforesting in accumulating parameters via type-directed transformations. *Asian Workshop on Programming Languages and Systems, Informal Proceedings*, pages 145–159.

[Nishimura, 2004] Fusion with stacks and accumulating parameters. *Partial Evaluation and Program Manipulation, Proceedings*, pages 101–112. ACM Press.

[Perst & Seidl, 2004] Macro forest transducers. *Information Processing Letters*, 89(3):141–149.

[Sands, 1996] Proving the correctness of recursion-based automatic program transformations. *Theoretical Computer Science*, 167(1–2):193–233.

[Svenningsson, 2002] Shortcut fusion for accumulating parameters & zip-like functions. *International Conference on Functional Programming, Proceedings*, SIGPLAN Notices 37(9):124–132. ACM Press.

[Wadler, 1990] Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248.

# Korrektheit auf unendlichen Bäumen (Abschnitt 6.1)

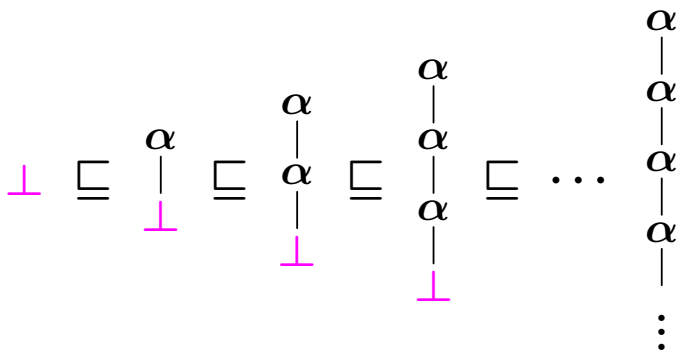
**Fragestellungen:**

1. **Was sind unendliche Bäume?**
2. **Wie rechnen Macro Tree Transducer auf diesen?**
3. **Sind Original- und komponiertes Programm äquivalent?**

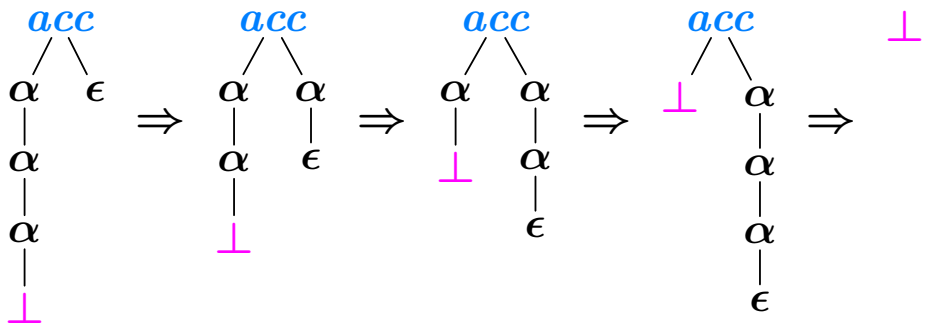
**Maßstab (für 2. und 3.): „operationelle Semantik“, z.B. Hugs**

# Der verfolgte Ansatz: „naive“ denotationelle Semantik

1. **Was sind unendliche Bäume?**




2. **Wie rechnen Macro Tree Transducer auf diesen?**
   **Per Limit der Ausgaben auf endlichen Approximationen:**



## Alternative: kategorielle Semantik (einer Teilsprache)?

**Ein Resultat von M. Barr (verallgemeinert durch J. Adámek):**

> **„Die finale Koalgebra zu einem bistetigen Endofunktor über der Kategorie der Mengen ist die Cauchy-Vervollständigung der initialen Algebra zu diesem Funktor.“**

**ermöglicht eine andere Sicht auf Punkt 1, denn:**

- **Polynomiale Funktoren über der Kategorie der Mengen erfüllen die notwendigen Voraussetzungen.**
- **Ihre initialen Algebren entsprechen Mengen endlicher Bäume.**
- **Ihre finalen Koalgebren entsprechen Mengen endlicher und unendlicher Bäume.**

# Cauchy-Vervollständigung / Begriffsbildung

**Metrik (auf $X$): Funktion $d : X \times X \longrightarrow \{r \in \mathbb{R} \mid r \geq 0\}$ mit**

- $d(x, y) = 0$ **gdw.** $x = y$
- $d(x, y) = d(y, x)$
- $d(x, y) + d(y, z) \geq d(x, z)$

**Cauchy-Folge:**
$x_0, x_1, x_2, \ldots$ **mit** $\forall \epsilon > 0.\ \exists n_0 \in \mathbb{N}.\ \forall n, m \geq n_0.\ d(x_n, x_m) \leq \epsilon$

**Konvergenz einer Folge $x_0, x_1, x_2, \ldots$ zu einem Limit $x$:**
$\forall \epsilon > 0.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ d(x_n, x) \leq \epsilon$

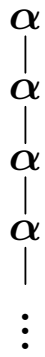
**Cauchy-Vollständigkeit: Jede Cauchy-Folge konvergiert.**

# Eine passende Metrik auf Bäumen

$$d(s,t) = \begin{cases} 0 & \text{falls } s = t \\ 2^{-k} & \text{sonst, für das größte } k \text{ so dass } s \text{ und } t \text{ bis Level } k \text{ identisch} \end{cases}$$

**Beispiel:**

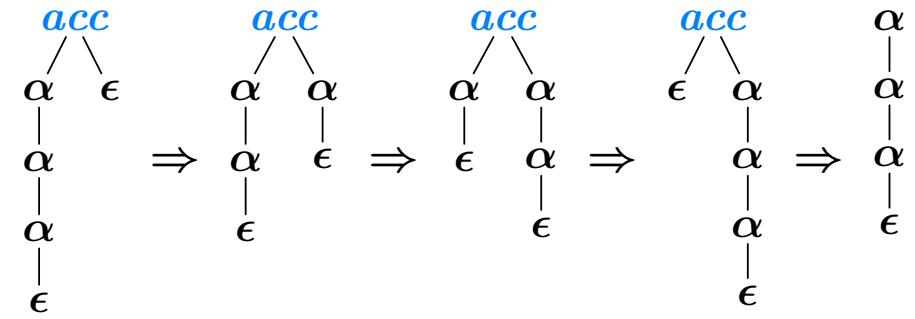
$\epsilon$ , $\alpha$ , $\alpha$ , $\alpha$ , $\cdots$ ist Cauchy und konvergiert zu



(scheinbarer) Vorteil: $\perp$ nicht notwendig

## Problematisch für Punkt 2 (und somit für Punkt 3):

Naheliegende Idee der Beschreibung der Ausgabe eines MTTs auf unendlicher Eingabe als Limit der Ausgaben auf Elementen einer passenden Cauchy-Folge scheitert:

Da 

und analog für andere endliche Eingaben, würde *acc* als Identität auch auf unendlicher Eingabe interpretiert. Dies entspricht jedoch nicht dem tatsächlichen Verhalten in der operationellen Referenzsemantik.

# Komplexität der Effizienzanalyse (1)

- **Tests auf Bedingungen aus Definition 2.5 sind mit linearem Aufwand in der Summe der Größen aller Regeln eines MTTs möglich.**
- **Für jedes $\phi \in RHS(F, \Delta \cup \{\circ, \bullet\}, U, Y)$ ist laut Theorem 5.8**

$$\exists \phi' \in RHS(F, \Delta \cup \{\bullet\}, U, Y).\ \phi \Rightarrow^*_{\mathcal{E}_{M_1, M_2}} \phi'$$

**äquivalent zu**

$$0 \leq sup_{\mathcal{E}_{M_1, M_2}}(\phi).$$

**Außerdem kann $sup_{\mathcal{E}_{M_1, M_2}}(\phi)$ entsprechend Definition 5.7 mit linearem Aufwand in der Größe von $\phi$ berechnet werden.**

$\Rightarrow$ **Tests auf Bedingungen aus Theoremen 4.20 und 4.47 sind mit linearem Aufwand in der Summe der Größen aller Regeln von $M_1$ und $M_2$ möglich.**

# Komplexität der Effizienzanalyse (2)

**Bei Theoremen 4.41 und 4.45:**

- **zusätzliche Quantifizierung**

$$\exists \kappa : \{(f,k) \mid f \in F^{(r+1)}, k \in [r]\} \longrightarrow \{0,\ldots,s_{max}\}.\ \ldots$$

$\Rightarrow$ $max(rank(G))^{1+\sum_{f \in F} rank_F(f)-1}$ **als zusätzlicher Faktor vor Größe der rechten Regelseiten von $M_1$ bei Zeitkomplexität**