Workshop GK 334, April 2, 2003

Efficiency Improvement by Tree Transducer Composition

Janis Voigtländer

Dresden University of Technology

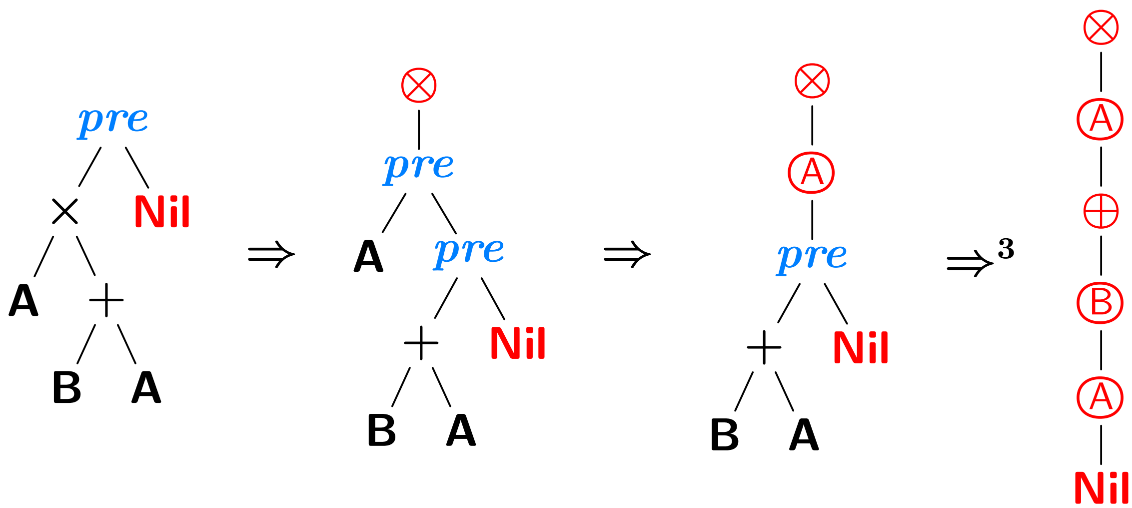
`http://wwwtcs.inf.tu-dresden.de/∼voigt`

Supported by the "Deutsche Forschungsgemeinschaft" under grants KU 1290/2-1 and KU 1290/2-3.

Macro Tree Transducers [Engelfriet, 1980]

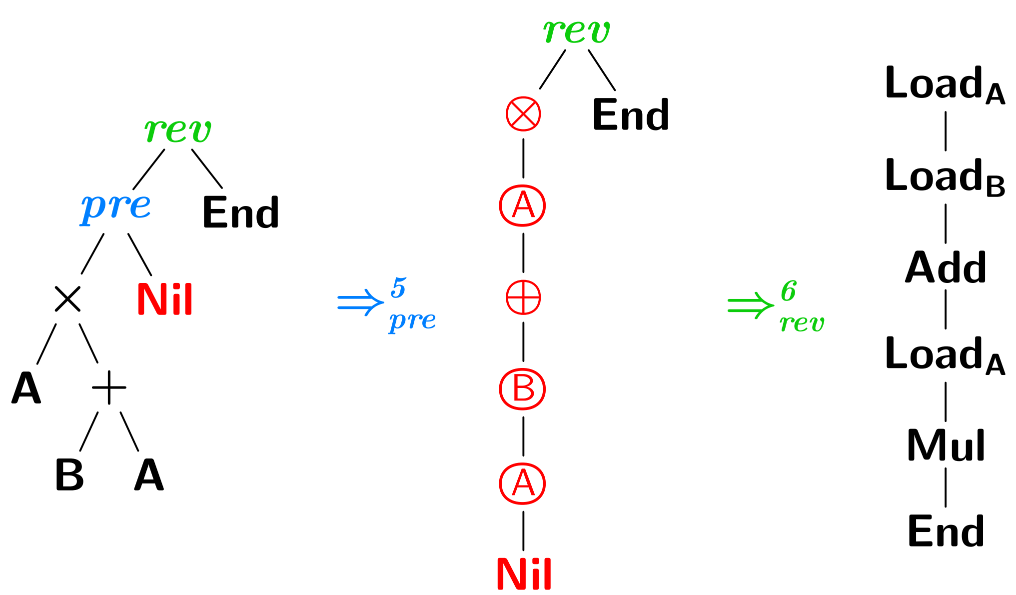
```
data Term = Term × Term | Term + Term | A | B
data List = ⊗ List | ⊕ List | Ⓐ List | Ⓑ List | Nil
data Ins  = Mul Ins | Add Ins | Load_A Ins | Load_B Ins | End

pre :: Term → List → List
pre (u1 × u2) y = ⊗ (pre u1 (pre u2 y))
pre (u1 + u2) y = ⊕ (pre u1 (pre u2 y))
pre    A      y = Ⓐ y
pre    B      y = Ⓑ y
```




```
rev :: List → Ins → Ins
rev (⊗ v) z = rev v (Mul z)
rev (⊕ v) z = rev v (Add z)
rev (Ⓐ v) z = rev v (Load_A z)
rev (Ⓑ v) z = rev v (Load_B z)
rev  Nil  z = z
main t = rev (pre t Nil) End
```

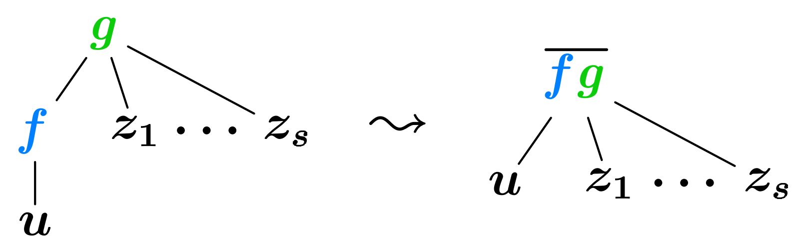
Modularity vs. Efficiency



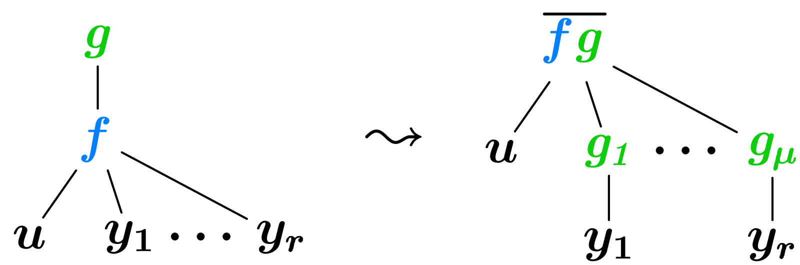
Deforestation techniques [Wadler, 1990; Gill *et al.*, 1993] fail to eliminate intermediate results inside accumulating parameters!

Composition Techniques for Tree Transducers

$\boldsymbol{TOP}$; $\boldsymbol{MAC} \subseteq \boldsymbol{MAC}$ **[Engelfriet, 1981]:**



$\boldsymbol{MAC}$; $\boldsymbol{TOP} \subseteq \boldsymbol{MAC}$ **[Engelfriet & Vogler, 1985]:**

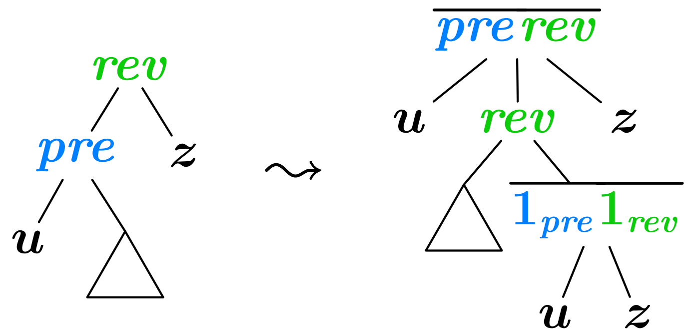


$\boldsymbol{MAC}_{su}$; $\boldsymbol{MAC}_{wsu} \subseteq \boldsymbol{MAC}$ **[Kühnemann, 1998]:**

$$\boldsymbol{MAC}_{su} ; \boldsymbol{MAC}_{wsu} \subseteq \boldsymbol{ATT}_{su} ; \boldsymbol{ATT} \subseteq \boldsymbol{ATT} \subseteq \boldsymbol{MAC}$$

Generalized Construction [V., 2001]

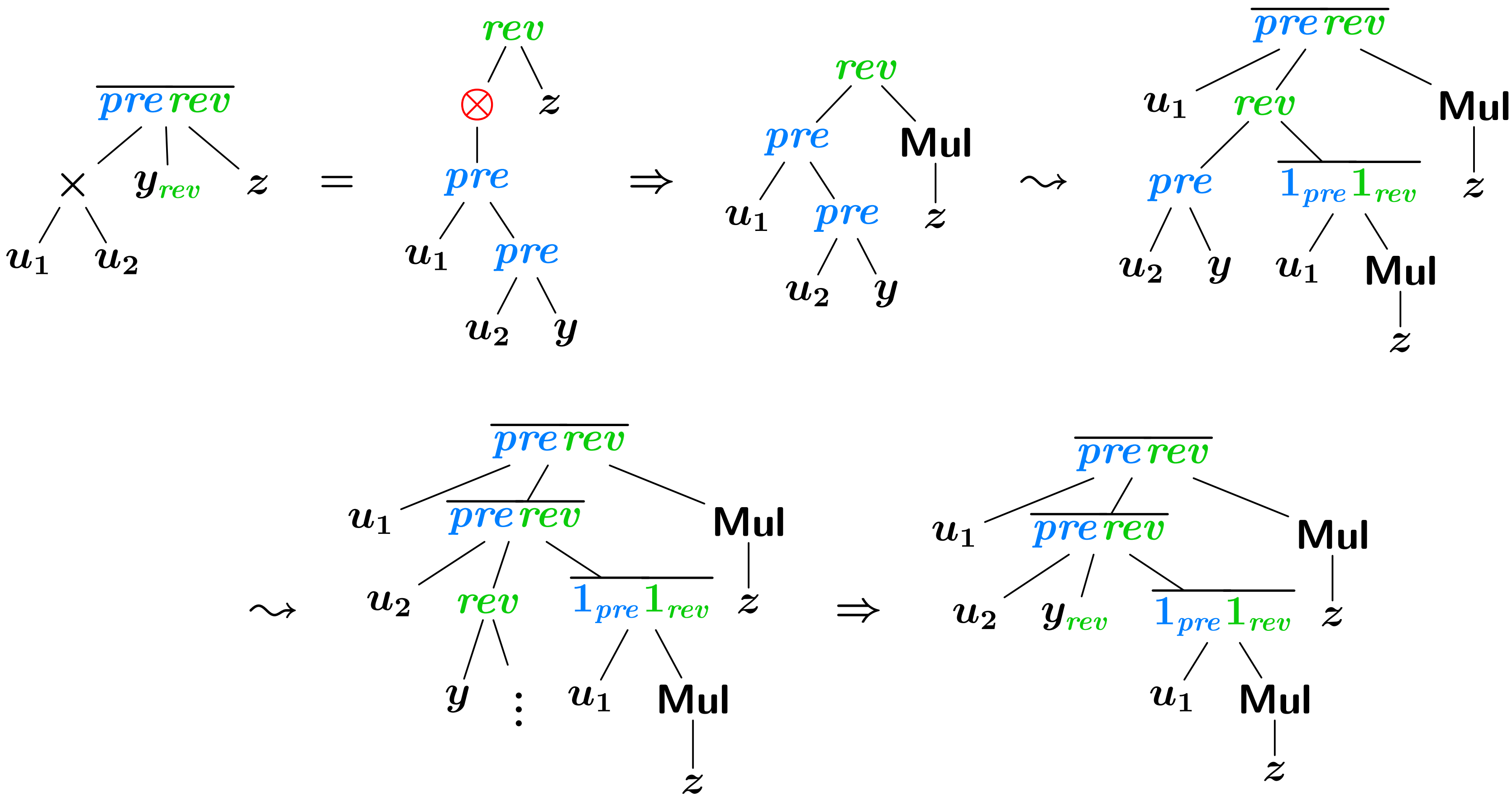
Replace compositions involving an intermediate result as follows:



Rules of $\overline{pre\,rev}$: obtained by translating right-hand sides of pre with rules of rev.

Rules of $\overline{1_{pre}1_{rev}}$: obtained by "walking upwards" in right-hand sides of pre.

Translating Right-Hand Sides: Example



Transformed Program

$$
\begin{array}{l}
\overline{prerev} :: \mathsf{Term} \rightarrow \mathsf{Ins} \rightarrow \mathsf{Ins} \rightarrow \mathsf{Ins} \\
\overline{prerev}\ (u_1 \times u_2)\ y_{rev}\ z = \overline{prerev}\ u_1\ (\overline{prerev}\ u_2\ y_{rev}\ (\overline{1_{pre}1_{rev}}\ u_1\ (\mathsf{Mul}\ z)))\ (\mathsf{Mul}\ z) \\
\overline{prerev}\ (u_1 + u_2)\ y_{rev}\ z = \overline{prerev}\ u_1\ (\overline{prerev}\ u_2\ y_{rev}\ (\overline{1_{pre}1_{rev}}\ u_1\ (\mathsf{Add}\ z)))\ (\mathsf{Add}\ z) \\
\overline{prerev}\ \mathsf{A}\ y_{rev}\ z = y_{rev} \\
\overline{prerev}\ \mathsf{B}\ y_{rev}\ z = y_{rev} \\
\\
\overline{1_{pre}1_{rev}} :: \mathsf{Term} \rightarrow \mathsf{Ins} \rightarrow \mathsf{Ins} \\
\overline{1_{pre}1_{rev}}\ (u_1 \times u_2)\ z = \overline{1_{pre}1_{rev}}\ u_2\ (\overline{1_{pre}1_{rev}}\ u_1\ (\mathsf{Mul}\ z)) \\
\overline{1_{pre}1_{rev}}\ (u_1 + u_2)\ z = \overline{1_{pre}1_{rev}}\ u_2\ (\overline{1_{pre}1_{rev}}\ u_1\ (\mathsf{Add}\ z)) \\
\overline{1_{pre}1_{rev}}\ \mathsf{A}\ z = \mathsf{Load}_{\mathsf{A}}\ z \\
\overline{1_{pre}1_{rev}}\ \mathsf{B}\ z = \mathsf{Load}_{\mathsf{B}}\ z \\
\\
main'\ t = \overline{prerev}\ t\ (\overline{1_{pre}1_{rev}}\ t\ \mathsf{End})\ \mathsf{End}
\end{array}
$$

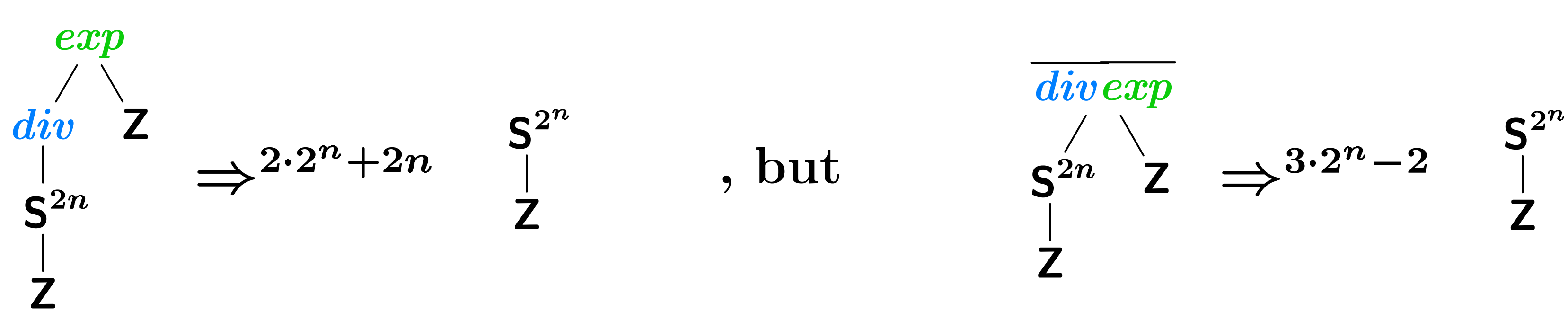
How does efficiency of this program relate to the original one?

Possible Loss of Efficiency

$$
\begin{aligned}
&\text{data } \mathsf{Nat} = \mathsf{S}\ \mathsf{Nat} \mid \mathsf{Z}\\
&div, div' :: \mathsf{Nat} \to \mathsf{Nat}\\
&div\ (\mathsf{S}\ u) = div'\ u\\
&div\ \ \mathsf{Z} = \mathsf{Z}\\
&div'\ (\mathsf{S}\ u) = \mathsf{S}\ (div\ u)\\
&div'\ \ \mathsf{Z} = \mathsf{Z}\\
&exp :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}\\
&exp\ (\mathsf{S}\ v)\ z = exp\ v\ (exp\ v\ z)\\
&exp\ \ \mathsf{Z}\ \ z = \mathsf{S}\ z\\
&main\ t = exp\ (div\ t)\ \mathsf{Z}
\end{aligned}
$$

$\leadsto$

$$
\begin{aligned}
&\overline{div\,exp}, \overline{div'\,exp} :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}\\
&\overline{div\,exp}\ (\mathsf{S}\ u)\ z = \overline{div'\,exp}\ u\ z\\
&\overline{div\,exp}\ \ \mathsf{Z}\ \ z = \mathsf{S}\ z\\
&\overline{div'\,exp}\ (\mathsf{S}\ u)\ z = \overline{div\,exp}\ u\ (\overline{div\,exp}\ u\ z)\\
&\overline{div'\,exp}\ \ \mathsf{Z}\ \ z = \mathsf{S}\ z\\
&main'\ t = \overline{div\,exp}\ t\ \mathsf{Z}
\end{aligned}
$$



Efficiency Analysis for Non-strict Evaluation is Difficult

$$
\begin{array}{lcl}
\mathit{isort}\ [] & = & [] \\
\mathit{isort}\ (x : xs) & = & \mathit{insert}\ x\ (\mathit{isort}\ xs) \\
\mathit{insert}\ x\ [] & = & [x] \\
\mathit{insert}\ x\ (y : ys) & = & \text{if } x \leq y \text{ then } x : y : ys \\
 & & \qquad\quad \text{else } \ y : \mathit{insert}\ x\ ys
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{qsort}\ [] & = & [] \\
\mathit{qsort}\ (x : xs) & = & \mathit{qsort}\ (\mathit{filter}\ (\leq x)\ xs) \\
 & & +\!\!+\ x : \mathit{qsort}\ (\mathit{filter}\ (> x)\ xs)
\end{array}
$$

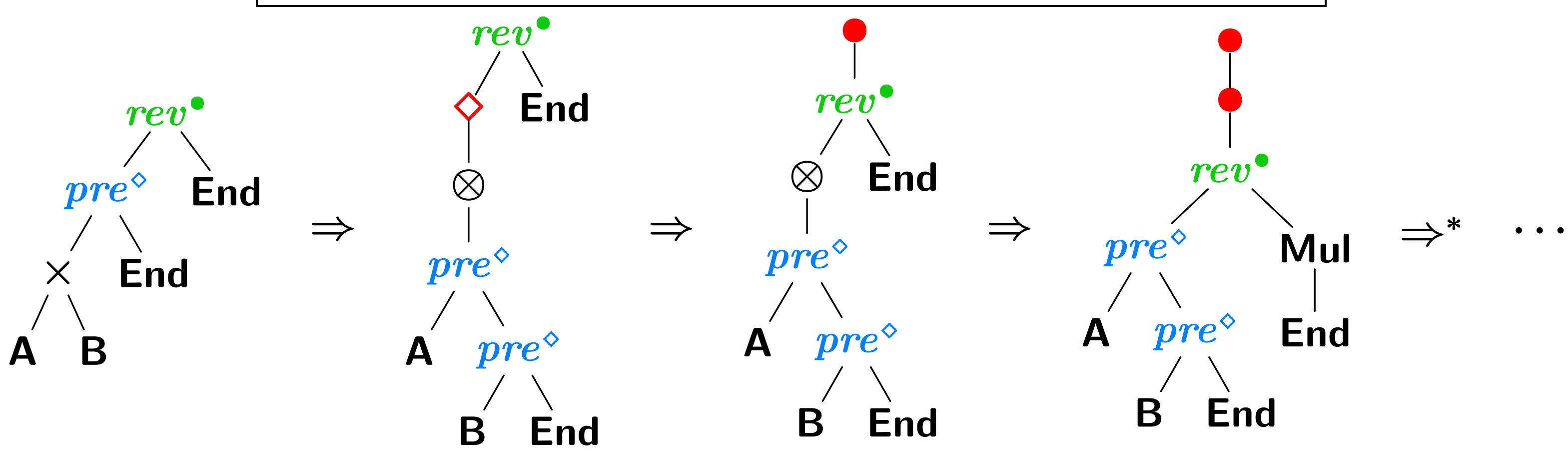
$$
\begin{array}{lcl}
\mathit{head}\ (x : xs) & = & x \\
\mathit{minimum}\ xs & = & \mathit{head}\ (\mathit{isort}\ xs)
\end{array}
$$

- *isort* and *qsort* require quadratic worst-case complexity, but *qsort* is more efficient in average case
- *minimum* has linear worst-case complexity, but replacing *qsort* for *isort* would make it quadratic

“Ticking” of Original Program

$$
\begin{aligned}
\mathit{pre}^{\diamond}\ (u_1 \times u_2)\ y &= \diamond\,(\otimes\,(\mathit{pre}^{\diamond}\ u_1\ (\mathit{pre}^{\diamond}\ u_2\ y)))\\
\mathit{pre}^{\diamond}\ (u_1 + u_2)\ y &= \diamond\,(\oplus\,(\mathit{pre}^{\diamond}\ u_1\ (\mathit{pre}^{\diamond}\ u_2\ y)))\\
\mathit{pre}^{\diamond}\quad \mathsf{A}\quad y &= \diamond\,(\text{Ⓐ}\ y)\\
\mathit{pre}^{\diamond}\quad \mathsf{B}\quad y &= \diamond\,(\text{Ⓑ}\ y)
\end{aligned}
$$

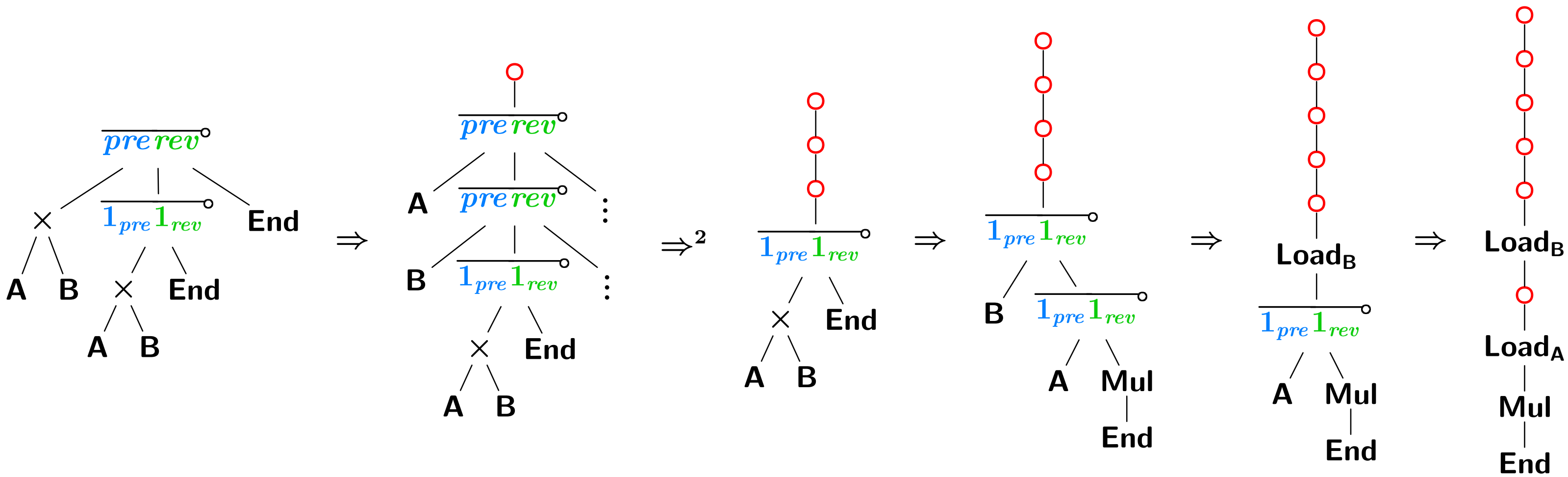
$$
\begin{aligned}
\mathit{rev}^{\bullet}\ (\diamond\ v)\ z &= \bullet\,(\mathit{rev}^{\bullet}\ v\ z)\\
\mathit{rev}^{\bullet}\ (\otimes\ v)\ z &= \bullet\,(\mathit{rev}^{\bullet}\ v\ (\mathsf{Mul}\ z))\\
\mathit{rev}^{\bullet}\ (\oplus\ v)\ z &= \bullet\,(\mathit{rev}^{\bullet}\ v\ (\mathsf{Add}\ z))\\
\mathit{rev}^{\bullet}\ (\text{Ⓐ}\ v)\ z &= \bullet\,(\mathit{rev}^{\bullet}\ v\ (\mathsf{Load_A}\ z))\\
\mathit{rev}^{\bullet}\ (\text{Ⓑ}\ v)\ z &= \bullet\,(\mathit{rev}^{\bullet}\ v\ (\mathsf{Load_B}\ z))\\
\mathit{rev}^{\bullet}\quad \mathsf{Nil}\quad z &= \bullet\ z
\end{aligned}
$$



Ticking of Composed Program

$$\overline{prerev}^{\circ}\ (u_1 \times u_2)\ y\ z = \circ\ (\overline{prerev}^{\circ}\ u_1\ (\overline{prerev}^{\circ}\ u_2\ y\ (\overline{1_{pre}1_{rev}}^{\circ}\ u_1\ (\mathsf{Mul}\ z)))\ (\mathsf{Mul}\ z))$$
$$\overline{prerev}^{\circ}\ (u_1 + u_2)\ y\ z = \circ\ (\overline{prerev}^{\circ}\ u_1\ (\overline{prerev}^{\circ}\ u_2\ y\ (\overline{1_{pre}1_{rev}}^{\circ}\ u_1\ (\mathsf{Add}\ z)))\ (\mathsf{Add}\ z))$$
$$\overline{prerev}^{\circ}\ \mathsf{A}\ y\ z = \circ\ y$$
$$\overline{prerev}^{\circ}\ \mathsf{B}\ y\ z = \circ\ y$$
$$\overline{1_{pre}1_{rev}}^{\circ}\ (u_1 \times u_2)\ z = \circ\ (\overline{1_{pre}1_{rev}}^{\circ}\ u_2\ (\overline{1_{pre}1_{rev}}^{\circ}\ u_1\ (\mathsf{Mul}\ z)))$$
$$\overline{1_{pre}1_{rev}}^{\circ}\ (u_1 + u_2)\ z = \circ\ (\overline{1_{pre}1_{rev}}^{\circ}\ u_2\ (\overline{1_{pre}1_{rev}}^{\circ}\ u_1\ (\mathsf{Add}\ z)))$$
$$\overline{1_{pre}1_{rev}}^{\circ}\ \mathsf{A}\ z = \circ\ (\mathsf{Load}_{\mathsf{A}}\ z)$$
$$\overline{1_{pre}1_{rev}}^{\circ}\ \mathsf{B}\ z = \circ\ (\mathsf{Load}_{\mathsf{B}}\ z)$$

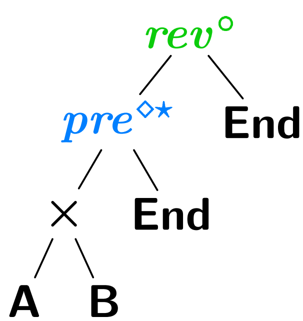
$$main'^{\circ}\ t = \overline{prerev}^{\circ}\ t\ (\overline{1_{pre}1_{rev}}^{\circ}\ t\ \mathsf{End})\ \mathsf{End}$$



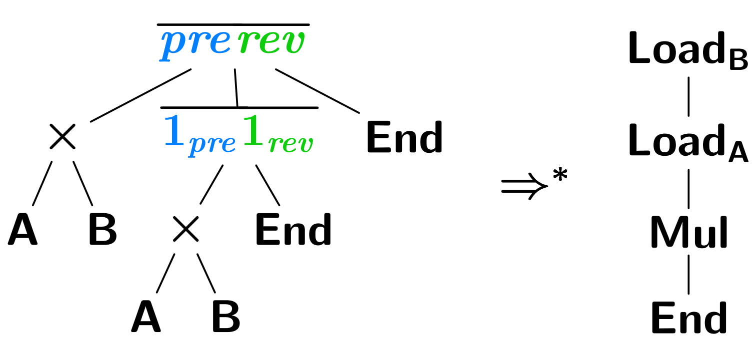
Annotation through Composition

$$
\begin{aligned}
pre^{\diamond\star}\ (u_1 \times u_2)\ y &= \diamond\ (\otimes\ (pre^{\diamond\star}\ u_1\ (pre^{\diamond\star}\ u_2\ (\star\ y))))\\
pre^{\diamond\star}\ (u_1 + u_2)\ y &= \diamond\ (\oplus\ (pre^{\diamond\star}\ u_1\ (pre^{\diamond\star}\ u_2\ (\star\ y))))\\
pre^{\diamond\star}\ \ \mathsf{A}\ \ y &= \diamond\ (Ⓐ\ (\star\ y))\\
pre^{\diamond\star}\ \ \mathsf{B}\ \ y &= \diamond\ (Ⓑ\ (\star\ y))\\
rev^{\circ}\ (\diamond\ v)\ z &= \circ\ (rev^{\circ}\ v\ z)\\
rev^{\circ}\ (\star\ v)\ z &= rev^{\circ}\ v\ (\circ\ z)\\
rev^{\circ}\ (\otimes\ v)\ z &= rev^{\circ}\ v\ (\mathsf{Mul}\ z)\\
rev^{\circ}\ (\oplus\ v)\ z &= rev^{\circ}\ v\ (\mathsf{Add}\ z)\\
rev^{\circ}\ (Ⓐ\ v)\ z &= rev^{\circ}\ v\ (\mathsf{Load_A}\ z)\\
rev^{\circ}\ (Ⓑ\ v)\ z &= rev^{\circ}\ v\ (\mathsf{Load_B}\ z)\\
rev^{\circ}\ \ \mathsf{Nil}\ \ z &= z
\end{aligned}
$$

Composes into the same program, hence the number of $\circ$-symbols in the reduction *result* of:



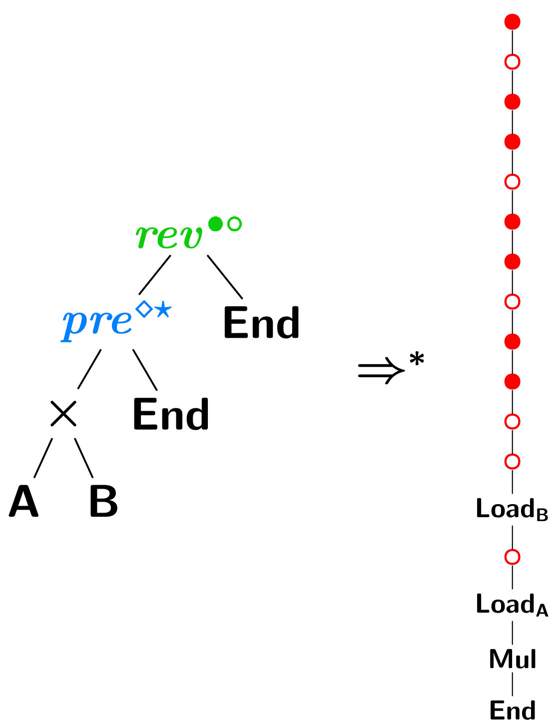
is equal to the number of call-by-name reduction *steps* of:



Combining Annotations

$$
\begin{aligned}
pre^{\diamond\star}\ (u_1 \times u_2)\ y &= \diamond\ (\otimes\ (pre^{\diamond\star}\ u_1\ (pre^{\diamond\star}\ u_2\ (\star\ y))))\\
pre^{\diamond\star}\ (u_1 + u_2)\ y &= \diamond\ (\oplus\ (pre^{\diamond\star}\ u_1\ (pre^{\diamond\star}\ u_2\ (\star\ y))))\\
pre^{\diamond\star}\ \mathsf{A}\ y &= \diamond\ (Ⓐ\ (\star\ y))\\
pre^{\diamond\star}\ \mathsf{B}\ y &= \diamond\ (Ⓑ\ (\star\ y))\\
rev^{\bullet\circ}\ (\diamond\ v)\ z &= \bullet\ (\circ\ (rev^{\bullet\circ}\ v\ z))\\
rev^{\bullet\circ}\ (\star\ v)\ z &= rev^{\bullet\circ}\ v\ (\circ\ z)\\
rev^{\bullet\circ}\ (\otimes\ v)\ z &= \bullet\ (rev^{\bullet\circ}\ v\ (\mathsf{Mul}\ z))\\
rev^{\bullet\circ}\ (\oplus\ v)\ z &= \bullet\ (rev^{\bullet\circ}\ v\ (\mathsf{Add}\ z))\\
rev^{\bullet\circ}\ (Ⓐ\ v)\ z &= \bullet\ (rev^{\bullet\circ}\ v\ (\mathsf{Load_A}\ z))\\
rev^{\bullet\circ}\ (Ⓑ\ v)\ z &= \bullet\ (rev^{\bullet\circ}\ v\ (\mathsf{Load_B}\ z))\\
rev^{\bullet\circ}\ \mathsf{Nil}\ z &= \bullet\ z
\end{aligned}
$$

Relative efficiency of original vs. transformed program can be determined by comparing numbers of $\bullet$- and $\circ$-symbols produced by above program:

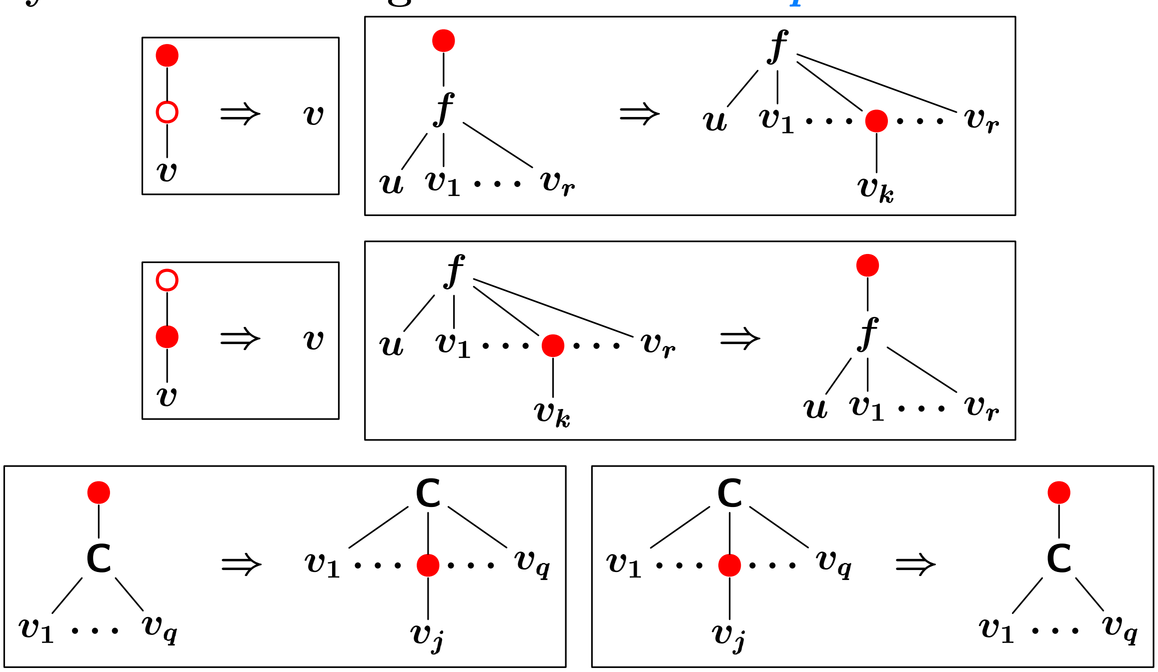


An Example Criterion at Work

$$
\begin{aligned}
pre^{\bullet\circ}\ (u_1 \times u_2)\ y &= \bullet\,(\otimes\,(pre^{\bullet\circ}\ u_1\ (pre^{\bullet\circ}\ u_2\ (\circ\ y))))\\
pre^{\bullet\circ}\ (u_1 + u_2)\ y &= \bullet\,(\oplus\,(pre^{\bullet\circ}\ u_1\ (pre^{\bullet\circ}\ u_2\ (\circ\ y))))\\
pre^{\bullet\circ}\ \mathsf{A}\ y &= \bullet\,(Ⓐ\,(\circ\ y))\\
pre^{\bullet\circ}\ \mathsf{B}\ y &= \bullet\,(Ⓑ\,(\circ\ y))
\end{aligned}
$$

$$
\begin{aligned}
rev^{\circ\bullet}\ (\circ\ v)\ z &= \circ\,(rev^{\circ\bullet}\ v\ z)\\
rev^{\circ\bullet}\ (\otimes\ v)\ z &= rev^{\circ\bullet}\ v\ (\mathsf{Mul}\ z)\\
rev^{\circ\bullet}\ (\oplus\ v)\ z &= rev^{\circ\bullet}\ v\ (\mathsf{Add}\ z)\\
rev^{\circ\bullet}\ (Ⓐ\ v)\ z &= rev^{\circ\bullet}\ v\ (\mathsf{Load_A}\ z)\\
rev^{\circ\bullet}\ (Ⓑ\ v)\ z &= rev^{\circ\bullet}\ v\ (\mathsf{Load_B}\ z)\\
rev^{\circ\bullet}\ \mathsf{Nil}\ z &= z\\
rev^{\circ\bullet}\ (\bullet\ v)\ z &= \bullet\,(rev^{\circ\bullet}\ v\ z)
\end{aligned}
$$

Since $pre^{\bullet\circ}$ is *context-linear* and *-nondeleting*, and $rev^{\circ\bullet}$ is *linear* and *nondeleting*, the following rules may be used with the aim of eliminating all $\circ$-symbols in the right-hand sides of $pre^{\bullet\circ}$:



Why Category Theory does not prove my Theorems

- free monads capture induction on tree structure, but we also do induction proofs on:
  - prefix order of paths
  - reversed subset order over sets of pairs (state,variable)
- finding (generalized) induction hypotheses is the really tough job; any support?
- not just translation of right-hand sides, but, e.g., "walking upwards"
- free monads do not count (as would be needed to characterize linearity restrictions, and in efficiency analysis)

References

[Engelfriet, 1980] Some open questions and recent results on tree transducers and tree languages. *In: Formal language theory; perspectives and open problems*. Academic Press.

[Engelfriet, 1981] *Tree transducers and syntax directed semantics*. Tech. rept. 363. Technische Hogeschool Twente.

[Engelfriet & Vogler, 1985] Macro tree transducers. *J. Comput. Syst. Sci.*, **31**, 71–145.

[Gill, Launchbury & Peyton Jones, 1993] A short cut to deforestation. *In: Functional Programming Languages and Computer Architecture, Copenhagen, Denmark*. ACM Press.

[Kühnemann, 1998] Benefits of tree transducers for optimizing functional programs. *In: Foundations of Software Technology & Theoretical Computer Science, Chennai, India*. LNCS, vol. 1530.

[Voigtländer, 2001] *Composition of restricted macro tree transducers*. M.Sc. thesis, Dresden University of Technology.

[Voigtländer, 2002] Conditions for efficiency improvement by tree transducer composition. *In: Rewriting Techniques and Applications, Copenhagen, Denmark*. LNCS, vol. 2378.

[Voigtländer & Kühnemann, 200?] Composition of functions with accumulating parameters. *J. Funct. Prog.,* to appear.

[Wadler, 1990] Deforestation: Transforming programs to eliminate trees. *Theoret. Comput. Sci.*, **73**, 231–248.