RTA, July 23, 2002

Conditions for Efficiency Improvement by Tree Transducer Composition

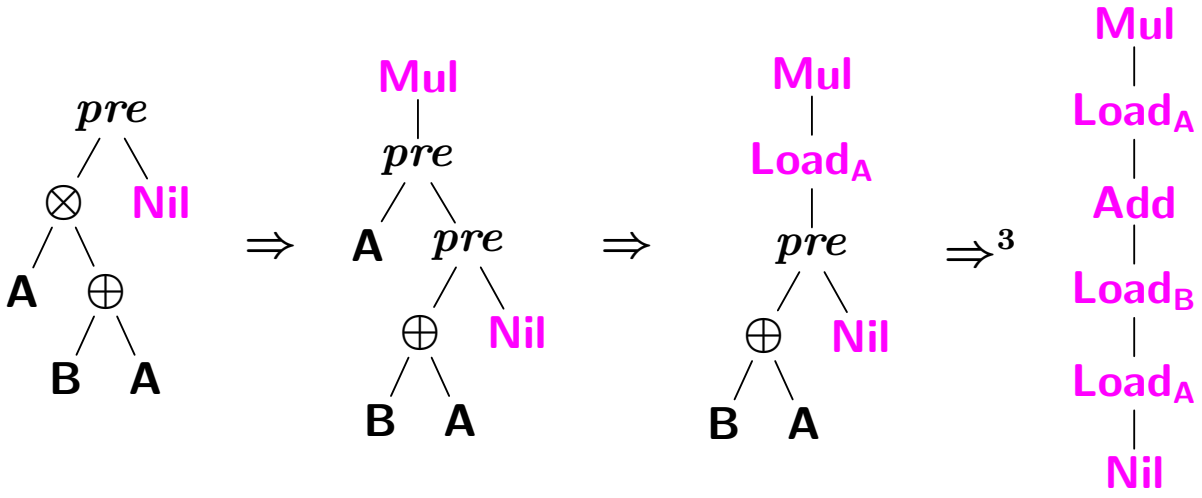
Janis Voigtländer
Dresden University of Technology
voigt@tcs.inf.tu-dresden.de

Supported by the "Deutsche Forschungsgemeinschaft" under grant KU 1290/2-1 and by the "European Commission – DG Information Society" with a FLoC'02 travel grant.

Macro Tree Transducers [Eng80]

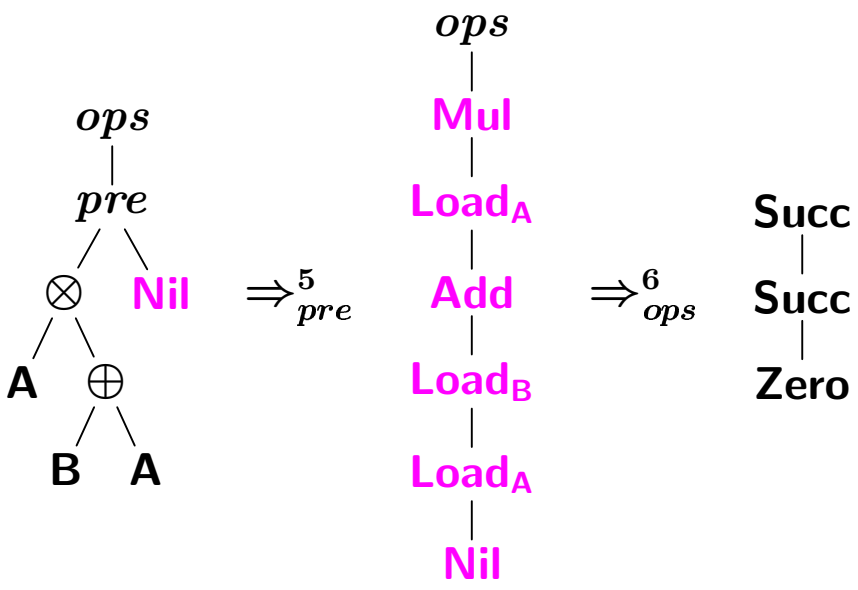
$$
\begin{array}{lcl}
\text{data } \mathsf{Term} & = & \mathsf{Term} \otimes \mathsf{Term} \mid \mathsf{Term} \oplus \mathsf{Term} \mid \mathsf{A} \mid \mathsf{B} \\
\text{data } \mathsf{Ins} & = & \mathsf{Mul\ Ins} \mid \mathsf{Add\ Ins} \mid \mathsf{Load_A\ Ins} \mid \mathsf{Load_B\ Ins} \mid \mathsf{Nil} \\
\text{data } \mathsf{Nat} & = & \mathsf{Succ\ Nat} \mid \mathsf{Zero}
\end{array}
$$

$$
\begin{array}{lcl}
pre :: \mathsf{Term} \to \mathsf{Ins} \to \mathsf{Ins} & & \\
pre\ (x_1 \otimes x_2)\ y & = & \mathsf{Mul}\ (pre\ x_1\ (pre\ x_2\ y)) \\
pre\ (x_1 \oplus x_2)\ y & = & \mathsf{Add}\ (pre\ x_1\ (pre\ x_2\ y)) \\
pre\ \ \mathsf{A}\ \ y & = & \mathsf{Load_A}\ y \\
pre\ \ \mathsf{B}\ \ y & = & \mathsf{Load_B}\ y
\end{array}
$$



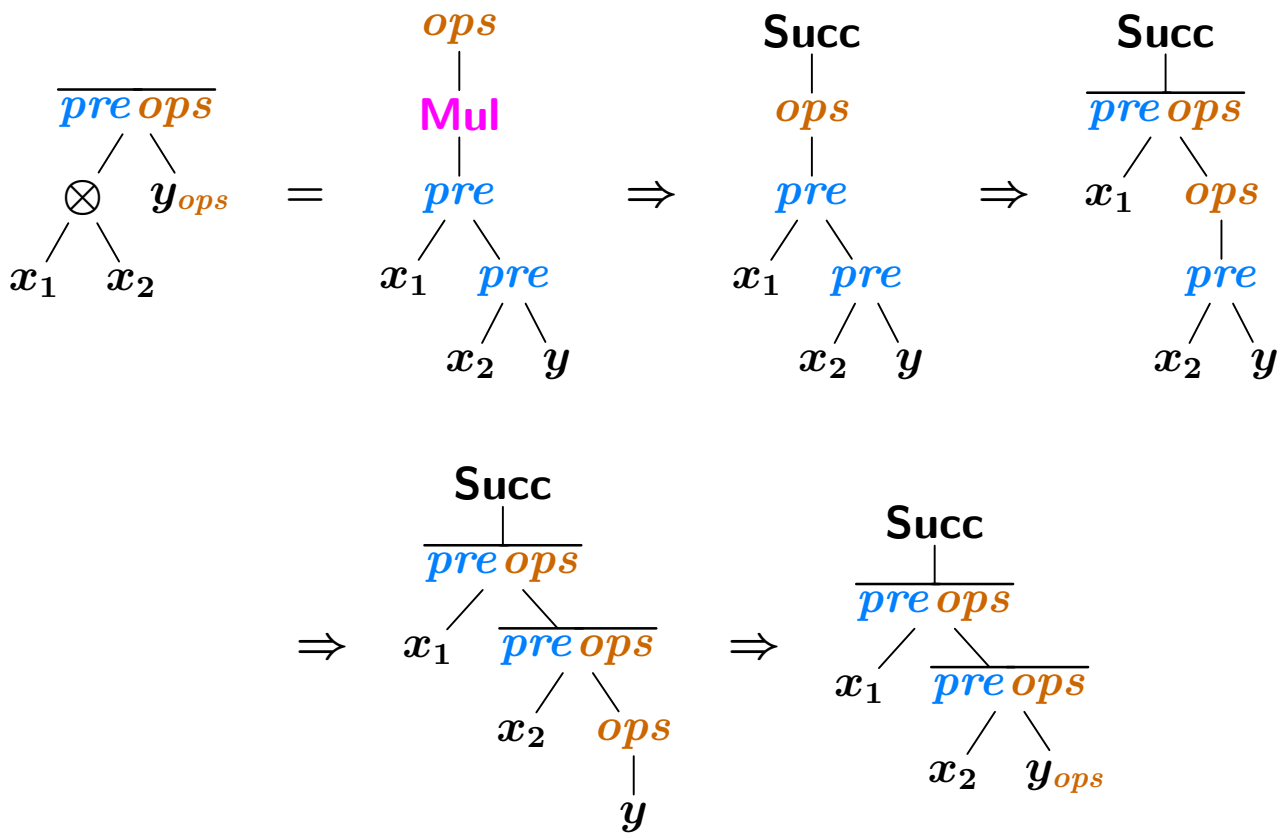
$$
\begin{array}{lcl}
ops :: \mathsf{Ins} \to \mathsf{Nat} & & \\
ops\ \ (\mathsf{Mul}\ x) & = & \mathsf{Succ}\ (ops\ x) \\
ops\ \ (\mathsf{Add}\ x) & = & \mathsf{Succ}\ (ops\ x) \\
ops\ (\mathsf{Load_A}\ x) & = & ops\ x \\
ops\ (\mathsf{Load_B}\ x) & = & ops\ x \\
ops\ \ \ \mathsf{Nil} & = & \mathsf{Zero}
\end{array}
$$

Intermediate Results



Inefficient !

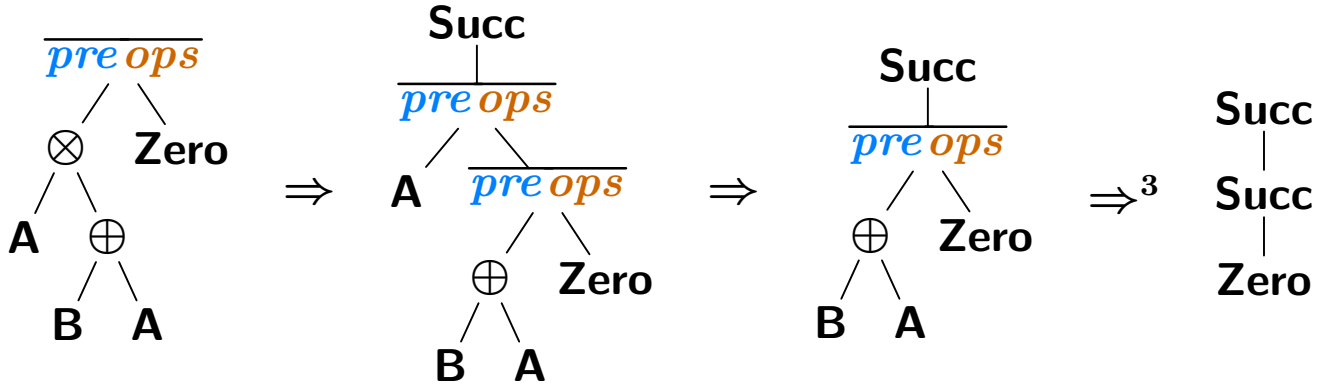
Tree Transducer Composition [EV85]: Example



Replace occurrences of (*ops* (*pre* *t* Nil)) by ($\overline{\textit{pre}\,\textit{ops}}$ *t* (*ops* Nil)).

Transformed Program:

$$
\begin{array}{lll}
\overline{pre\,ops} :: \mathsf{Term} \rightarrow \mathsf{Nat} \rightarrow \mathsf{Nat} & & \\
\overline{pre\,ops}\ (x_1 \otimes x_2)\ y_{ops} & = & \mathsf{Succ}\ (\overline{pre\,ops}\ x_1\ (\overline{pre\,ops}\ x_2\ y_{ops})) \\
\overline{pre\,ops}\ (x_1 \oplus x_2)\ y_{ops} & = & \mathsf{Succ}\ (\overline{pre\,ops}\ x_1\ (\overline{pre\,ops}\ x_2\ y_{ops})) \\
\overline{pre\,ops}\ \ \mathsf{A}\ \ y_{ops} & = & y_{ops} \\
\overline{pre\,ops}\ \ \mathsf{B}\ \ y_{ops} & = & y_{ops}
\end{array}
$$



No intermediate result is produced !

Transformed program requires fewer reduction steps !

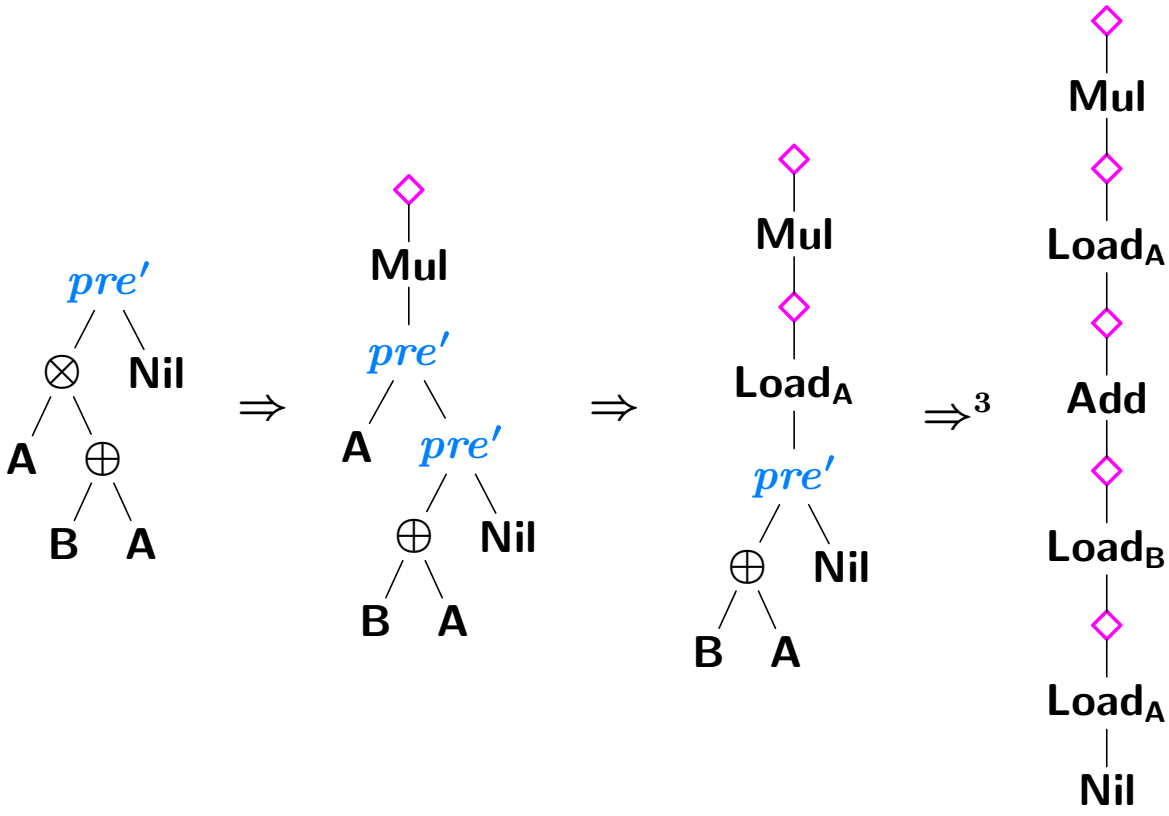
Formal Efficiency Analysis

should be:

- **with respect to call-by-need reduction steps**
- **input-independent**
- **based on original program before transformation**

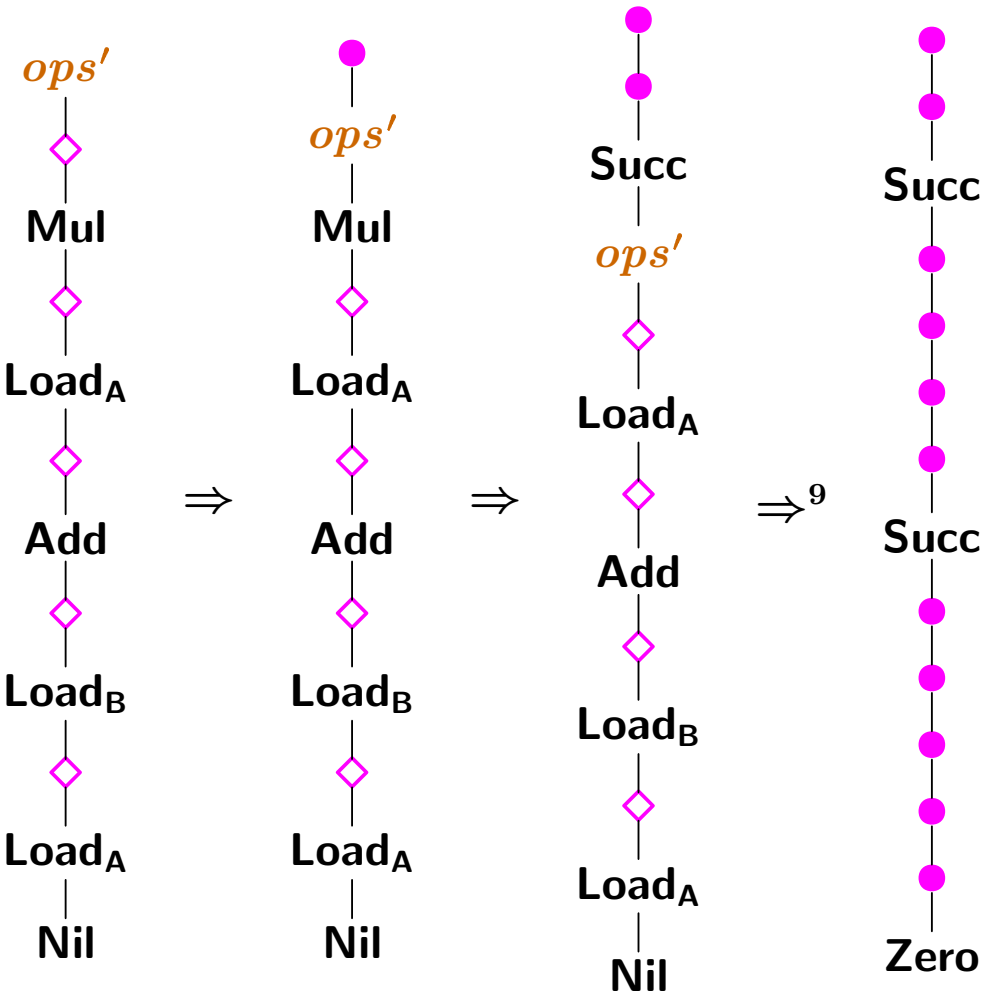
Ticking of Producer:

$$
\begin{array}{lllll}
pre'\ (x_1 \otimes x_2)\ y & = & \diamond\ (\mathsf{Mul}\ (pre'\ x_1\ (pre'\ x_2\ y))) \\
pre'\ (x_1 \oplus x_2)\ y & = & \diamond\ (\mathsf{Add}\ (pre'\ x_1\ (pre'\ x_2\ y))) \\
pre'\ \quad \mathsf{A} \quad y & = & \diamond\ (\mathsf{Load_A}\ y) \\
pre'\ \quad \mathsf{B} \quad y & = & \diamond\ (\mathsf{Load_B}\ y)
\end{array}
$$



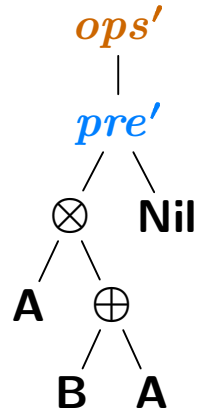
Ticking of Consumer:

$$
\begin{array}{lcl}
ops'\ (\mathsf{Mul}\ x) & = & \bullet\ (\mathsf{Succ}\ (ops'\ x)) \\
ops'\ (\mathsf{Add}\ x) & = & \bullet\ (\mathsf{Succ}\ (ops'\ x)) \\
ops'\ (\mathsf{Load_A}\ x) & = & \bullet\ (ops'\ x) \\
ops'\ (\mathsf{Load_B}\ x) & = & \bullet\ (ops'\ x) \\
ops'\ \mathsf{Nil} & = & \bullet\ \mathsf{Zero} \\
ops'\ (\diamond\ x_1) & = & \bullet\ (ops'\ x_1)
\end{array}
$$

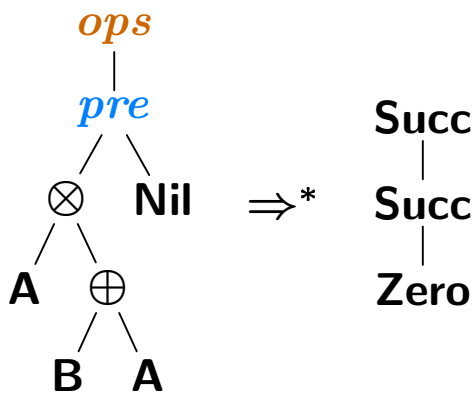


Steps of Original Program Reflected in Output (Lemma 2)

The number of •-symbols in the reduction *result* of:

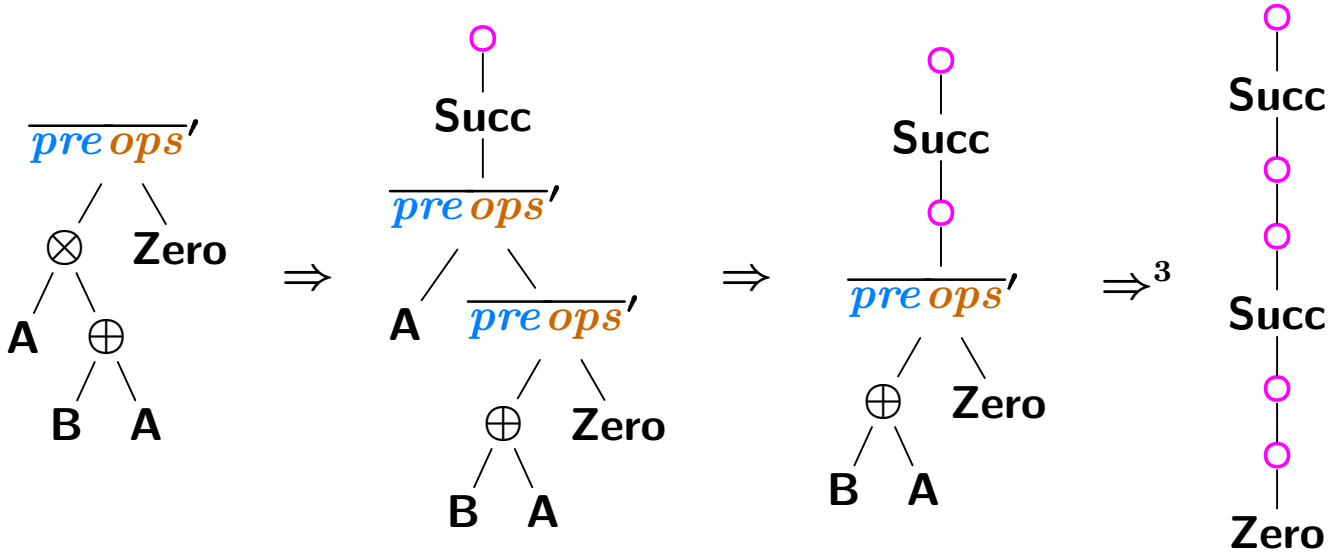


is equal to the number of call-by-need reduction *steps* of:



Ticking of Composed Program:

$$
\begin{array}{lll}
\overline{pre\,ops}'\ (x_1 \otimes x_2)\ y_{ops} & = & \circ\ (\mathsf{Succ}\ (\overline{pre\,ops}'\ x_1\ (\overline{pre\,ops}'\ x_2\ y_{ops}))) \\
\overline{pre\,ops}'\ (x_1 \oplus x_2)\ y_{ops} & = & \circ\ (\mathsf{Succ}\ (\overline{pre\,ops}'\ x_1\ (\overline{pre\,ops}'\ x_2\ y_{ops}))) \\
\overline{pre\,ops}'\ \ \mathsf{A}\ \ y_{ops} & = & \circ\ y_{ops} \\
\overline{pre\,ops}'\ \ \mathsf{B}\ \ y_{ops} & = & \circ\ y_{ops}
\end{array}
$$



Annotation through Composition (Lemma 4)

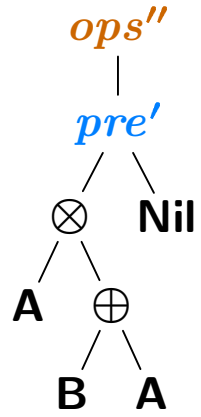
$$
\begin{array}{lcl}
pre'\ (x_1 \otimes x_2)\ y & = & \diamond\ (\mathsf{Mul}\ (pre'\ x_1\ (pre'\ x_2\ y))) \\
pre'\ (x_1 \oplus x_2)\ y & = & \diamond\ (\mathsf{Add}\ (pre'\ x_1\ (pre'\ x_2\ y))) \\
pre'\ \ \mathsf{A}\ \ y & = & \diamond\ (\mathsf{Load_A}\ y) \\
pre'\ \ \mathsf{B}\ \ y & = & \diamond\ (\mathsf{Load_B}\ y) \\
\\
ops''\ \ (\mathsf{Mul}\ x) & = & \mathsf{Succ}\ (ops''\ x) \\
ops''\ \ (\mathsf{Add}\ x) & = & \mathsf{Succ}\ (ops''\ x) \\
ops''\ (\mathsf{Load_A}\ x) & = & ops''\ x \\
ops''\ (\mathsf{Load_B}\ x) & = & ops''\ x \\
ops''\ \ \mathsf{Nil} & = & \mathsf{Zero} \\
ops''\ \ (\diamond\ x_1) & = & \circ\ (ops''\ x_1)
\end{array}
$$

composed into:

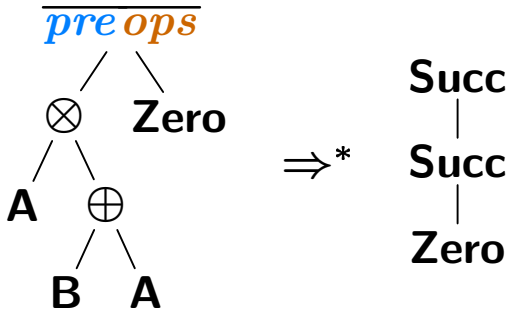
$$
\begin{array}{lcl}
\overline{pre'\,ops''}\ (x_1 \otimes x_2)\ y_{ops''} & = & \circ\ (\mathsf{Succ}\ (\overline{pre'\,ops''}\ x_1\ (\overline{pre'\,ops''}\ x_2\ y_{ops''}))) \\
\overline{pre'\,ops''}\ (x_1 \oplus x_2)\ y_{ops''} & = & \circ\ (\mathsf{Succ}\ (\overline{pre'\,ops''}\ x_1\ (\overline{pre'\,ops''}\ x_2\ y_{ops''}))) \\
\overline{pre'\,ops''}\ \ \mathsf{A}\ \ y_{ops''} & = & \circ\ y_{ops''} \\
\overline{pre'\,ops''}\ \ \mathsf{B}\ \ y_{ops''} & = & \circ\ y_{ops''}
\end{array}
$$

Steps of Composed Program Reflected in Output (Lemma 5)

The number of ∘-symbols in the reduction *result* of:



is greater or equal to the number of call-by-need reduction *steps* of:



Compare Annotated Programs:

$$
\begin{aligned}
pre'\ (x_1 \otimes x_2)\ y &= \diamond\ (\mathsf{Mul}\ (pre'\ x_1\ (pre'\ x_2\ y))) \\
pre'\ (x_1 \oplus x_2)\ y &= \diamond\ (\mathsf{Add}\ (pre'\ x_1\ (pre'\ x_2\ y))) \\
pre'\quad \mathsf{A}\quad y &= \diamond\ (\mathsf{Load_A}\ y) \\
pre'\quad \mathsf{B}\quad y &= \diamond\ (\mathsf{Load_B}\ y)
\end{aligned}
$$

$ops'\ (\mathsf{Mul}\ x) = \bullet\ (\mathsf{Succ}\ (ops'\ x))$	$ops''\ (\mathsf{Mul}\ x) = \mathsf{Succ}\ (ops''\ x)$
$ops'\ (\mathsf{Add}\ x) = \bullet\ (\mathsf{Succ}\ (ops'\ x))$	$ops''\ (\mathsf{Add}\ x) = \mathsf{Succ}\ (ops''\ x)$
$ops'\ (\mathsf{Load_A}\ x) = \bullet\ (ops'\ x)$	$ops''\ (\mathsf{Load_A}\ x) = ops''\ x$
$ops'\ (\mathsf{Load_B}\ x) = \bullet\ (ops'\ x)$	$ops''\ (\mathsf{Load_B}\ x) = ops''\ x$
$ops'\quad \mathsf{Nil} = \bullet\ \mathsf{Zero}$	$ops''\quad \mathsf{Nil} = \mathsf{Zero}$
$ops'\quad (\diamond\ x_1) = \bullet\ (ops'\ x_1)$	$ops''\quad (\diamond\ x_1) = \circ\ (ops''\ x_1)$

Always more $\bullet$- than $\circ$-symbols !

Abstracting from the Example (Theorem 1)

The composed program is at least as efficient as the original program, provided that:

1. **the producer is *context-linear* or *basic***
2. **the consumer is *recursion-linear***
3. **the consumer is *context-linear* or *basic***

Further Results:

- **Weaker pre-conditions by counting only steps of the consumer (Lemma 3, Lemma 6, Lemma 7, Theorem 2)**
- **Application to special cases of *classical deforestation* [Wad90] (Corollary 1)**
- **Analysis technique scales for the case that both involved transducers use context parameters [VK01]; work in progress**

References

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems*, pages 241–286. New York, Academic Press, 1980.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.

[VK01] J. Voigtländer and A. Kühnemann. Composition of functions with accumulating parameters. Technical Report TUD-FI01-08, Dresden University of Technology, August 2001.
http://wwwtcs.inf.tu-dresden.de/∼voigt/TUD-FI01-08.ps.gz.

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoret. Comput. Sci.*, 73:231–248, 1990.