Direct Construction and Efficiency Analysis for the Accumulation Technique for 2-modular Tree Transducers

Andreas Maletti

Department of Computer Science

Dresden University of Technology

TECHNISCHE UNIVERSITÄT DRESDEN

Itinerary

- Motivation and Introduction
- Modular Tree Transducers
- Basic Accumulation Technique
- Efficiency Considerations
- Extended Accumulation Technique
- Conclusions

Motivating Example

- **Inefficient Reverse:**

```
irev :: [Bool] -> [Bool]
irev (False : xs) = irev xs ++ [False]
irev (True  : xs) = irev xs ++ [True]
irev      []      = []
```

- **Efficient Reverse:**

```
rev :: [Bool] -> [Bool]
rev x = r x []
      where r :: [Bool] -> [Bool] -> [Bool]
            r (False : xs) ys = r xs (False : ys)
            r (True  : xs) ys = r xs (True  : ys)
            r      []      ys = ys
```

Remarks

Inefficient Reverse	Efficient Reverse
easy to comprehend	somewhat more complex
automatic verification possible	verification requires user
quadratic time complexity	linear time complexity
$(\frac{k^2+k}{2} + k + 1)$	$(k + 1)$

if k is the length of the input list.

$\longrightarrow$ write and prove the inefficient version, but run the efficient one.

Another Example

```
data Nat = S Nat | Z
```

- **Non-accumulating Fibonacci:**

```
nf, nf' :: Nat -> Nat
nf  (S x) = nf' x + nf x    |   nf' (S x) = nf x
nf    Z   = S Z             |   nf'   Z   = Z
```

- **Accumulating Fibonacci:**

```
fib :: Nat -> Nat
fib x = f x Z
      where f, f' :: Nat -> Nat -> Nat
            f  (S x) y = f' x (f x y)
            f    Z   y = S y
            f' (S x) y = f x y
            f'   Z   y = y
```

Remarks

Efficiency consideration for input $(S^n Z)$; $n \in \mathbb{N}$:

- **Non-accumulating Fibonacci:**

$$\mathrm{Fib}(n+1) + \mathrm{Fib}(n+3) + \sum_{i=0}^{n-2} (\mathrm{Fib}(i) * \mathrm{Fib}(n-2-i)) - 3$$

- **Accumulating Fibonacci:**

$$\mathrm{Fib}(n+3) - 2$$

Modular tree transducer

A *modular tree transducer* is a quintuple $M = (F, m, \Delta, e, R)$ where:

- F is a ranked alphabet of *function symbols* ($F^{(0)} = \emptyset$),
- $m : F \longrightarrow \mathbb{N}$ is the *module mapping*,
- Δ is a ranked alphabet of *constructors* disjoint to F,
- $e = (f\ x\ t_1\ \dots\ t_r)$ for some $f \in F^{(r+1)}$ and $t_1, \dots, t_r \in \mathcal{T}_\Delta$; $r \in \mathbb{N}$ and
- R – the set of *rewrite rules* – contains for every $k, r \in \mathbb{N}$, $f \in F^{(r+1)}$ and $\delta \in \Delta^{(k)}$ an equation

$$f\ (\delta\ x_1\ \dots\ x_k)\ y_1\ \dots\ y_r = \mathrm{rhs}_M(f, \delta)$$

Modular tree transducer (Example)

The 2-modular tree transducer M_{rev} is defined as

$$M_{\text{rev}} = (\{\text{rev}^{(1)}, \text{app}^{(2)}\}, m, \{A^{(1)}, B^{(1)}, N^{(0)}\}, (\text{rev}\ x), R)$$

with $m(\text{rev}) = 1$, $m(\text{app}) = 2$ and rule-set R :

$$\begin{array}{llll}
\text{rev} & N & & = & N \\
\text{rev} & (A\ x) & & = & \text{app}\ (\text{rev}\ x)\ (A\ N) \\
\text{rev} & (B\ x) & & = & \text{app}\ (\text{rev}\ x)\ (B\ N) \\
\text{app} & N & y & = & y \\
\text{app} & (A\ x) & y & = & A\ (\text{app}\ x\ y) \\
\text{app} & (B\ x) & y & = & B\ (\text{app}\ x\ y).
\end{array}$$

Properties of modules

- *Top-down tree transducer module:* a module consisting solely of unary function symbols
- *Substitution module:*
  - there exists $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\} \subseteq \Delta^{(0)}$; $\mathrm{mx} \in \mathbb{N}$ such that for every $1 \le i \le \mathrm{mx} : \mathrm{rhs}_M(\mathrm{sub}, \Pi_i) = y_i$ where $\mathrm{sub} \in F^{(\mathrm{mx}+1)}$ is the only function symbol of the module
  - "endomorphic extension" to constructors of $\Delta \setminus \Pi$

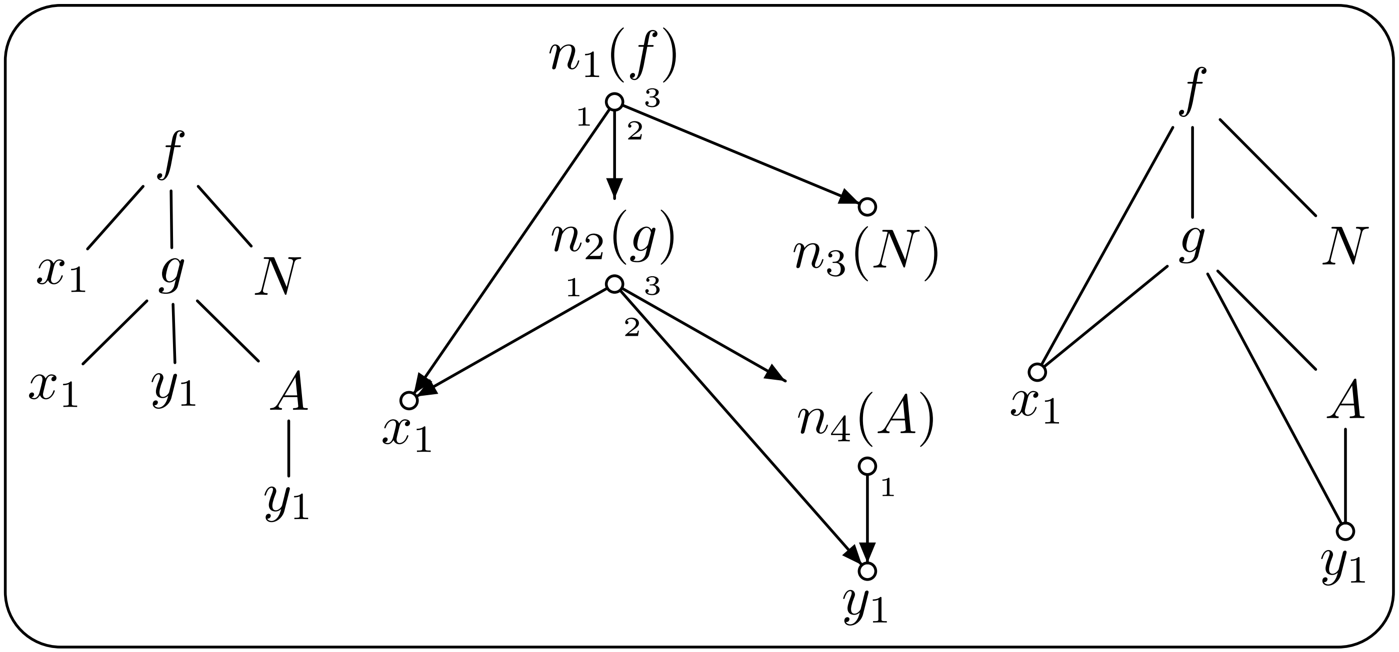
Term graphs

A *term graph* is a quadruple (N, E, l, r) where

- N is a set of nodes,
- E is a partial mapping $E : N \times \mathbb{N}_+ \longrightarrow N$ (successor mapping),
- l is a partial mapping $l : N \longrightarrow \Sigma$ (labelling mapping),
- $r \in N$ is the designated root node

and certain restrictions (non-circularity, connectedness, etc.) apply.

Term graph examples



Call-by-need

Ξ_1"$|$ => "$_M \Xi_2$, if and only if

1. **Locate redex:** there exists a least element $p \in R_M(\Xi_1)$ with respect to the lexicographic ordering on paths, which requires a term graph homomorphism ψ from $G_{\rho,\mathrm{lhs}}$ to $\Xi_1|_p$ for some $\rho \in R$,
2. **Build:** let $G_{\rho,\mathrm{lhs}} = (N_{\mathrm{lhs}}, E_{\mathrm{lhs}}, l_{\mathrm{lhs}}, r_{\mathrm{lhs}})$ and $G_{\rho,\mathrm{rhs}} = (N_{\mathrm{rhs}}, E_{\mathrm{rhs}}, l_{\mathrm{rhs}}, r_{\mathrm{rhs}})$ with $N_{\mathrm{rhs}} \cap N_{\Xi_1} = \emptyset$, then

$G' = (N', E', l', r_{\Xi_1})$ is defined as

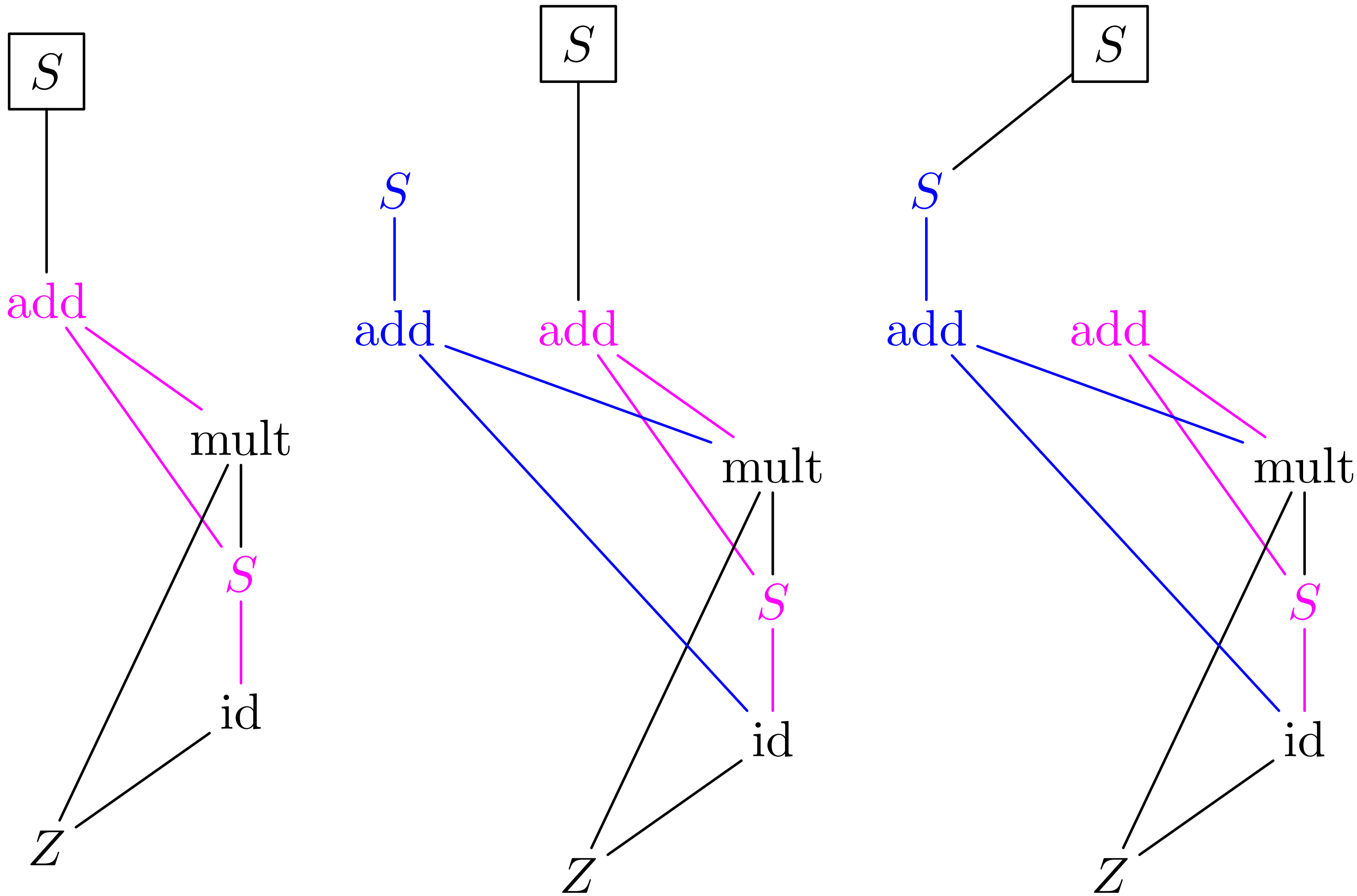
$$
\begin{aligned}
N' &= N_{\Xi_1} \cup (N_{\mathrm{rhs}} \setminus N_{\mathrm{lhs}}) \\
E'(n,i) &= \begin{cases} E_{\Xi_1}(n,i) & \text{, if } n \in N_{\Xi_1} \\ E_{\mathrm{rhs}}(n,i) & \text{, if } n, E_{\mathrm{rhs}}(n,i) \in N_{\mathrm{rhs}} \setminus N_{\mathrm{lhs}} \\ \psi(E_{\mathrm{rhs}}(n,i)) & \text{, otherwise} \end{cases} \\
l'(n) &= \begin{cases} l_{\Xi_1}(n) & \text{, if } n \in N_{\Xi_1} \\ l_{\mathrm{rhs}}(n) & \text{, otherwise} \end{cases}
\end{aligned}
$$

for every $n \in N'$ and $i \in \mathbb{N}_+$ and

3. **Redirect:** $\Xi_2 = G'[\psi(r_{\mathrm{lhs}}) \rightsquigarrow r_{\mathrm{rhs}}]$, where we ensure the connectedness property, i.e. trigger garbage collection, if necessary.

Steps illustrated

`add (S x) y = S (add x y)`



The construction

Given a 2-modular tree transducer $M = (F, m, \Delta, e, R)$, whose first module is a top-down tree transducer module and the second module is a substitution module with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$. Construct $M' = (F', \Delta, e', R')$

- $F' = \{f^{(\mathrm{mx}+1)} \mid f \in F,\ m(f) = 1\}$,
- $e' = (f\ x\ \Pi_1\ \ldots\ \Pi_{\mathrm{mx}})$, if $e = (f\ x)$, and
- the right hand sides at $f \in F'$ are constructed by translating the original right hand sides at f.

Translation

- $\mathrm{trans}(\Pi_j, \xi_1, \ldots, \xi_{\mathrm{mx}}) = \xi_j$
- $\mathrm{trans}((f\ x), \xi_1, \ldots, \xi_{\mathrm{mx}}) = (f\ x\ \xi_1\ \ldots\ \xi_{\mathrm{mx}})$
- $\mathrm{trans}((\delta\ s_1\ \ldots\ s_k), \xi_1, \ldots, \xi_{\mathrm{mx}}) =$

$$(\delta\ \mathrm{trans}(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})\ \ldots\ \mathrm{trans}(s_k, \xi_1, \ldots, \xi_{\mathrm{mx}}))$$

- $\mathrm{trans}((\mathrm{sub}\ s\ s_1\ \ldots\ s_{\mathrm{mx}}), \xi_1, \ldots, \xi_{\mathrm{mx}}) =$

$$\mathrm{trans}(s, \mathrm{trans}(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})\ \ldots\ \mathrm{trans}(s_{\mathrm{mx}}, \xi_1, \ldots, \xi_{\mathrm{mx}}))$$

Example

The right hand side of the rev function symbol at A is computed as follows:

$$\begin{aligned} & \text{trans}(\text{app}\ (\text{rev}\ x_1)\ (A\ \Pi_1), y_1) \\ = \ & \text{trans}((\text{rev}\ x_1), \text{trans}((A\ \Pi_1), y_1)) \\ = \ & \text{rev}\ x_1\ (\text{trans}((A\ \Pi_1), y_1)) \\ = \ & \text{rev}\ x_1\ (A\ (\text{trans}(\Pi_1, y_1))) \\ = \ & \text{rev}\ x_1\ (A\ y_1). \end{aligned}$$

Correctness proof sketch

- Lift trans to sentential forms,
- prove

$$\begin{array}{ccc} \xi & \xrightarrow{\text{trans}} & \text{trans}(\xi, \Pi_1, \ldots, \Pi_{\text{mx}}) \\ \Downarrow_M & & \Downarrow_{M'}^{*} \\ \xi' & \xrightarrow{\text{trans}} & \text{trans}(\xi', \Pi_1, \ldots, \Pi_{\text{mx}}) \end{array}$$

- since $\text{trans}(\xi, \Pi_1, \ldots, \Pi_{\text{mx}}) = \xi$, if $\xi \in \mathcal{T}_\Delta$, we gain the desired result.

Efficiency deterioration

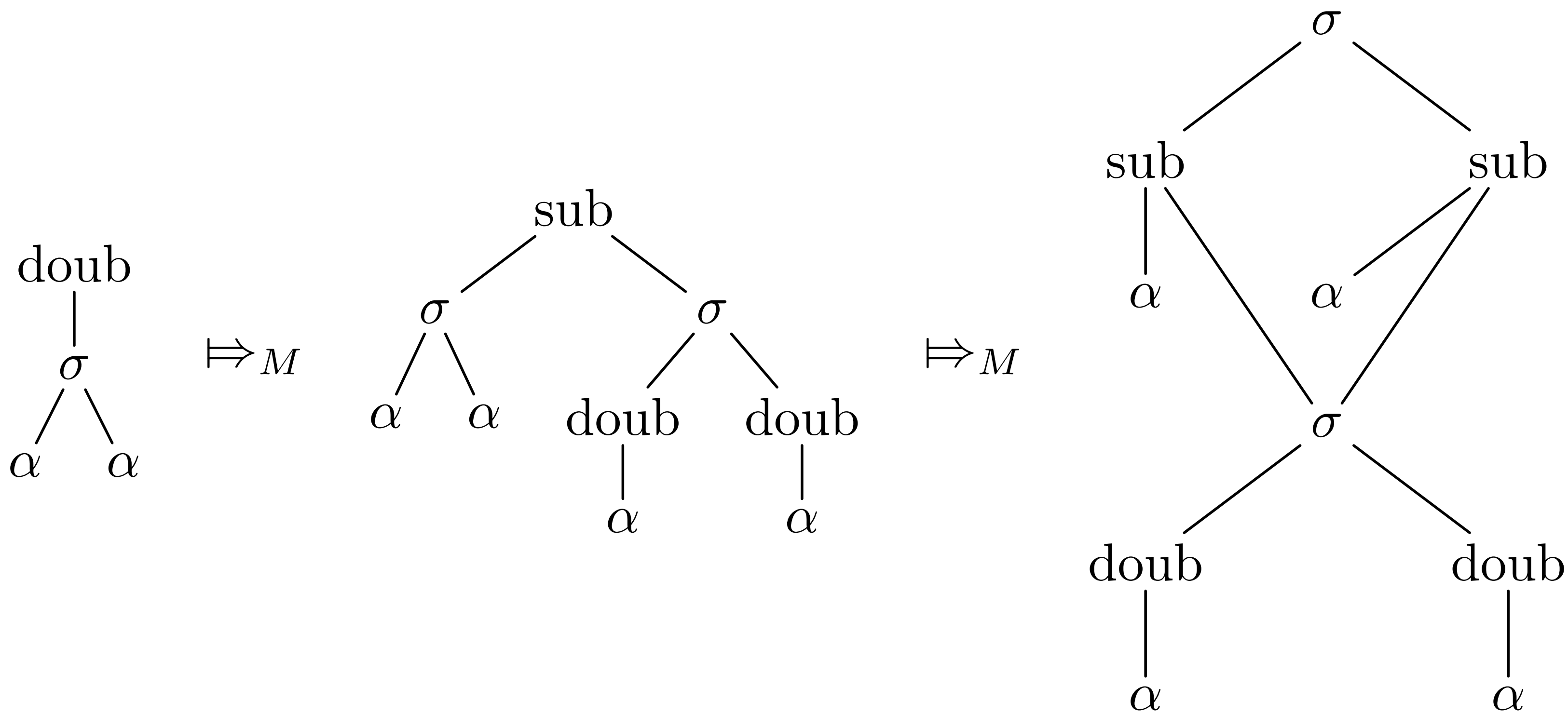
- Non-accumulating version M:

$$
\begin{aligned}
\text{doub}\ \ & \alpha & & = \alpha \\
\text{doub}\ \ & (\sigma\, x_1\, x_2) & & = \text{sub}\, (\sigma\, \alpha\, \alpha)\, (\sigma\, (\text{doub}\, x_1)\, (\text{doub}\, x_2)) \\
\text{sub}\ \ & \alpha & y_1 & = y_1 \\
\text{sub}\ \ & (\sigma\, x_1\, x_2) & y_1 & = \sigma\, (\text{sub}\, x_1\, y_1)\, (\text{sub}\, x_2\, y_1).
\end{aligned}
$$

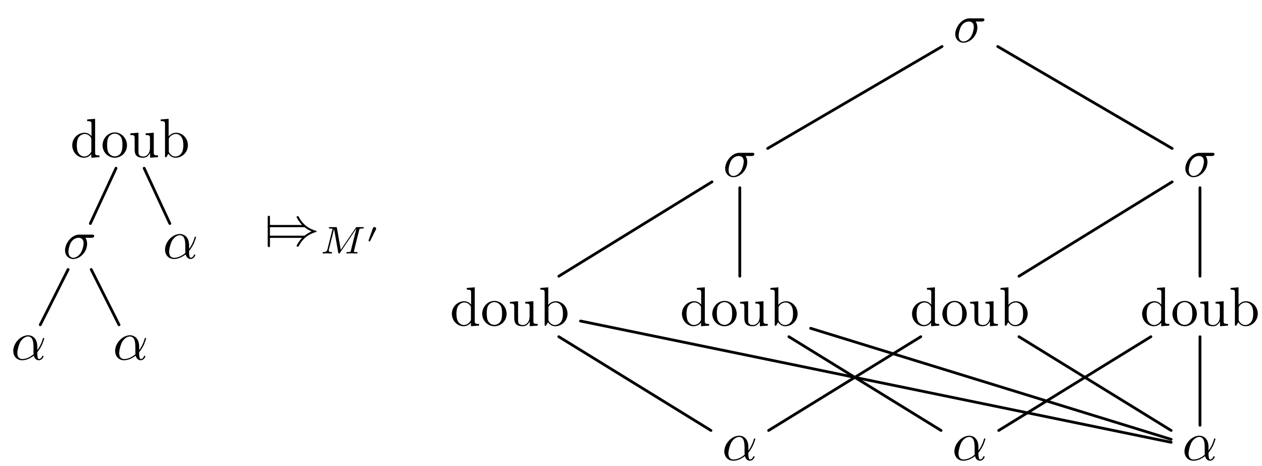
- Accumulating version M':

$$
\begin{aligned}
\text{doub}\ \ & \alpha & y_1 & = y_1 \\
\text{doub}\ \ & (\sigma\, x_1\, x_2) & y_1 & = \sigma\, (\sigma\, (\text{doub}\, x_1\, y_1)\, (\text{doub}\, x_2\, y_1)) \\
& & & \quad (\sigma\, (\text{doub}\, x_1\, y_1)\, (\text{doub}\, x_2\, y_1)).
\end{aligned}
$$

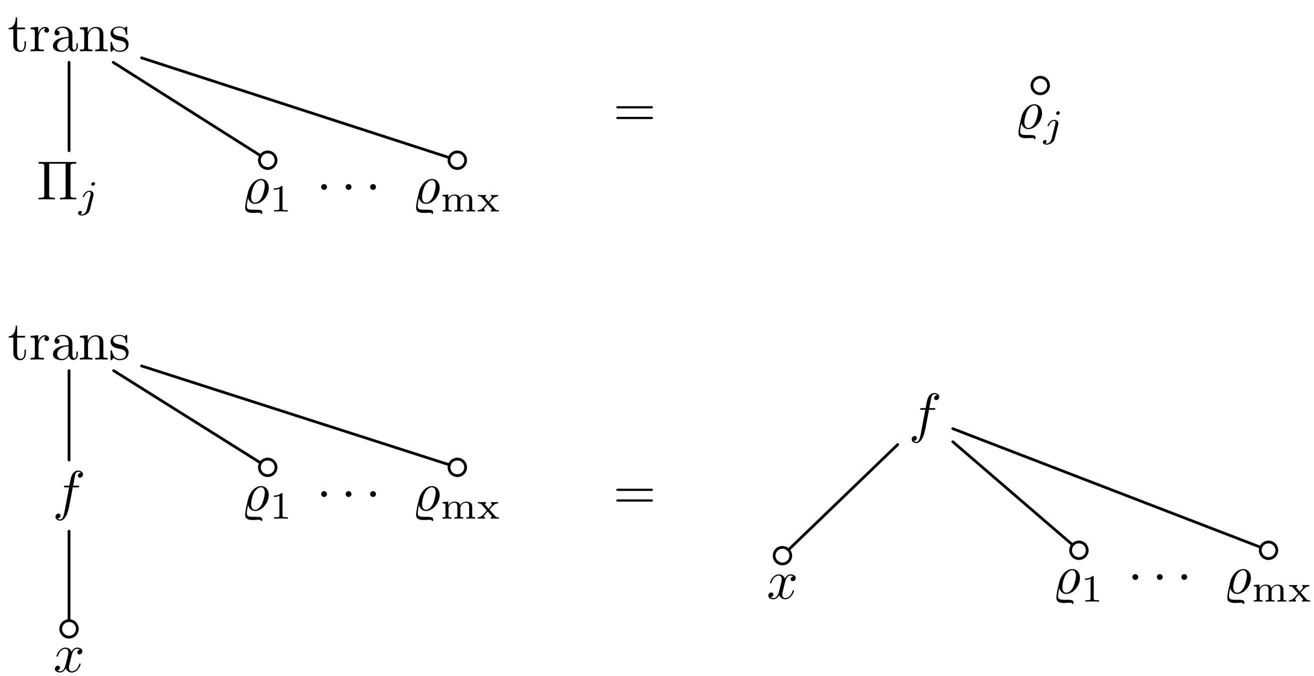
Derivation using M



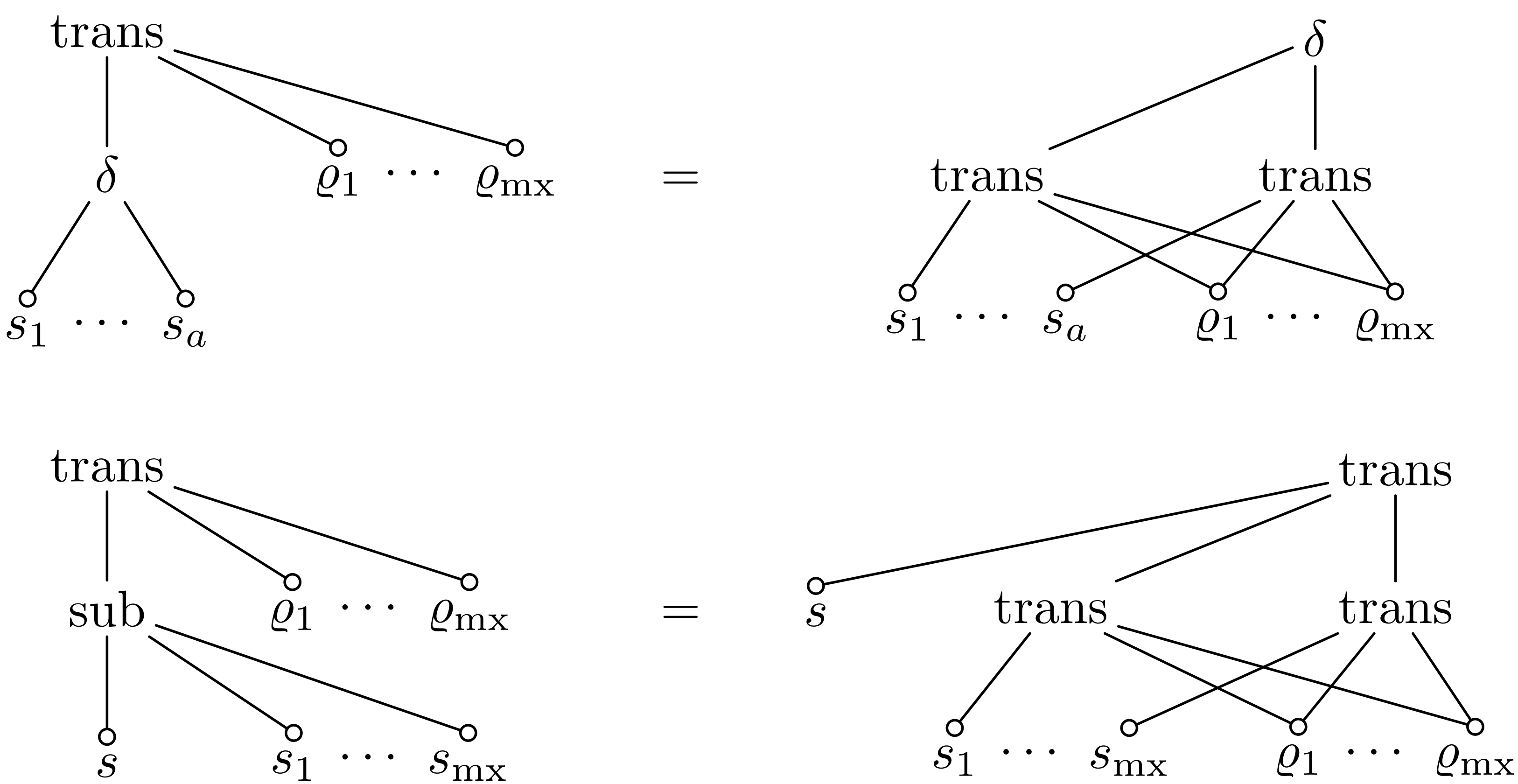
Derivation using M'



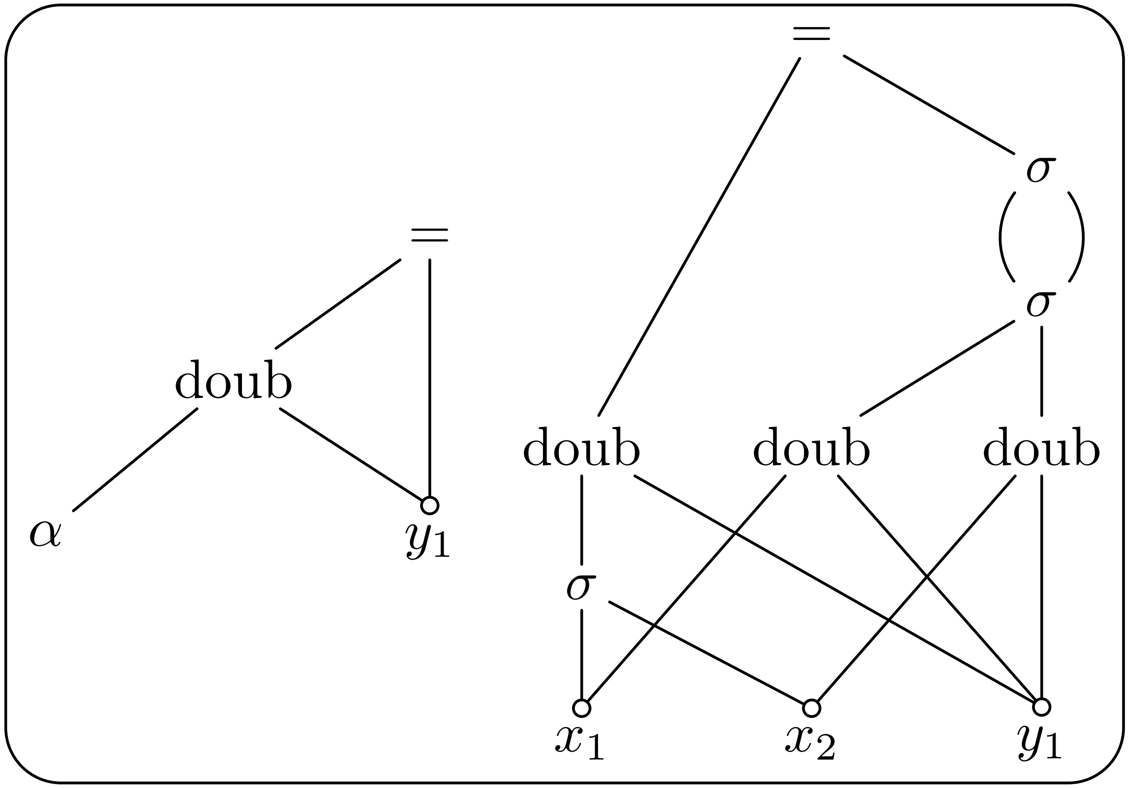
Refined translation



Refined translation (cont'd)



Example




```
doub      A      y = y
doub (S x1 x2) y = S z z
                     where z = S (doub x1 y) (doub x2 y)
```

Proof sketch of efficiency non-deterioration

- Lift trans to sentential forms,
- prove

$$\begin{array}{ccc} \xi & \xrightarrow{\text{trans}} & \text{trans}(\xi, \Pi_1, \ldots, \Pi_{\text{mx}}) \\ \Big\Downarrow_{M} & & \Big\Downarrow_{M'}^{0/1} \\ \xi' & \xrightarrow{\text{trans}} & \text{trans}(\xi', \Pi_1, \ldots, \Pi_{\text{mx}}) \end{array}$$

- Thereby, whenever M performs one derivation step, M' performs at most one derivation step. Since the call-by-need derivation relation is actually a partial mapping (deterministic), we gain the desired result.

Extended construction

- Non-accumulating version:

$$
\begin{array}{lcccl}
\text{coll} & N & y_1 & = & y_1 \\
\text{coll} & (\,|\,x_1) & y_1 & = & \text{app}\, y_1\, (\text{coll}\, x_1\, N) \\
\text{coll} & (R\, x_1) & y_1 & = & R\, (\text{coll}\, x_1\, y_1) \\
\text{coll} & (W\, x_1) & y_1 & = & \text{coll}\, x_1\, (W\, y_1) \\
\\
\text{app} & N & y_1 & = & y_1 \\
\text{app} & (\,|\,x_1) & y_1 & = & |\, (\text{app}\, x_1\, y_1) \\
\text{app} & (R\, x_1) & y_1 & = & R\, (\text{app}\, x_1\, y_1) \\
\text{app} & (W\, x_1) & y_1 & = & W\, (\text{app}\, x_1\, y_1).
\end{array}
$$

- Accumulating version:

$$
\begin{aligned}
&\text{coll}\ N && y_1\ z_1 = y_1\\
&\text{coll}\ (\,|\ x_1) && y_1\ z_1 = y_1\\
&\text{coll}\ (R\ x_1) && y_1\ z_1 = R\ (\text{coll}\ x_1\ y_1\ z_1)\\
&\text{coll}\ (W\ x_1) && y_1\ z_1 = \text{coll}\ x_1\ (W\ y_1)\ z_1
\end{aligned}
$$

$$
\begin{aligned}
&(1,1)\ N && y_1\ z_1 = z_1\\
&(1,1)\ (\,|\ x_1) && y_1\ z_1 = \text{coll}\ x_1\ ((1,1)\ x_1\ N\ z_1)\ z_1\\
&(1,1)\ (R\ x_1) && y_1\ z_1 = (1,1)\ x_1\ N\ z_1\\
&(1,1)\ (W\ x_1) && y_1\ z_1 = (1,1)\ x_1\ N\ z_1.
\end{aligned}
$$