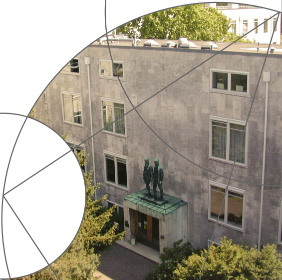
Programming Macro Tree Transducers

Patrick Bahr[1] Laurence E. Day[2]

[1]University of Copenhagen,
Department of Computer Science
paba@diku.dk

[2]University of Nottingham,
Functional Programming Laboratory
led@cs.nott.ac.uk



Agenda

1. String Acceptors & String Transducers

Agenda

1. String Acceptors & String Transducers
2. Tree Acceptors & Tree Transducers

Agenda

1. String Acceptors & String Transducers
2. Tree Acceptors & Tree Transducers
3. Programming with Tree Transducers in Haskell

Agenda

1. String Acceptors & String Transducers
2. Tree Acceptors & Tree Transducers
3. Programming with Tree Transducers in Haskell
4. Tree Transducers with Polymorphic State Space

Agenda

1. String Acceptors & String Transducers
2. Tree Acceptors & Tree Transducers
3. Programming with Tree Transducers in Haskell
4. Tree Transducers with Polymorphic State Space
5. Macro Tree Transducers
   (= Tree Transducers with Accumulation Parameters)

Agenda

1. String Acceptors & String Transducers
2. Tree Acceptors & Tree Transducers
3. Programming with Tree Transducers in Haskell
4. Tree Transducers with Polymorphic State Space
5. Macro Tree Transducers
   (= Tree Transducers with Accumulation Parameters)

Agenda

1. String Acceptors & String Transducers
2. Tree Acceptors & Tree Transducers
3. Programming with Tree Transducers in Haskell
4. Tree Transducers with Polymorphic State Space
5. Macro Tree Transducers
   (= Tree Transducers with Accumulation Parameters)

Why Tree Transducers?

- Compositionality $\rightsquigarrow$ deforestation
- Manipulation of transducers
- Composition with state transition functions

Finite State Automata – On Strings

w o r d

Finite State Automata – On Strings

q_0 w o r d

Finite State Automata – On Strings

q_0 w o r d

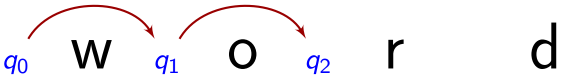
$$q, s \rightarrow q'$$

Finite State Automata – On Strings

q_0 w q_1 o r d

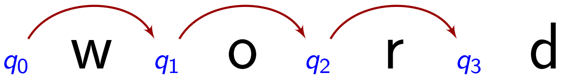
$$q, s \rightarrow q'$$

Finite State Automata – On Strings



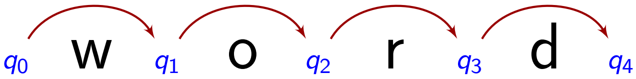
$$q, s \rightarrow q'$$

Finite State Automata – On Strings



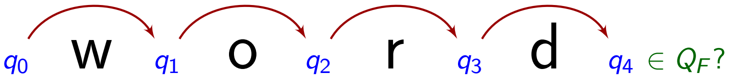
$$q, s \;\to\; q'$$

Finite State Automata – On Strings



$$q, s \rightarrow q'$$

Finite State Automata – On Strings



$$q, s \rightarrow q'$$

Finite State Automata – On Strings

q_0 w q_1 o q_2 r q_3 d $q_4 \in Q_F$?

Acceptor

$$q, s \rightarrow q'$$

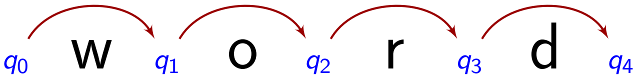
Finite State Automata – On Strings

q_0 w q_1 o q_2 r q_3 d $q_4 \in Q_F$?

Transducer?

$$q, s \quad \rightarrow \quad q'$$

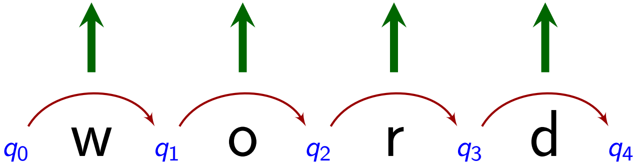
Finite State Automata – On Strings



Transducer

$$q, s \quad \rightarrow \quad q', w$$

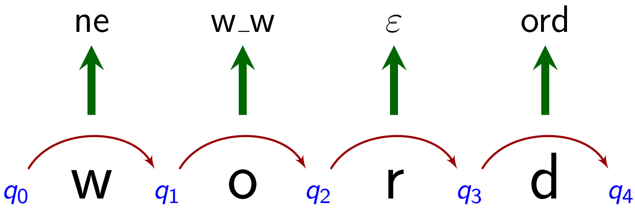
Finite State Automata – On Strings



Transducer

$$q, s \quad \rightarrow \quad q', w$$

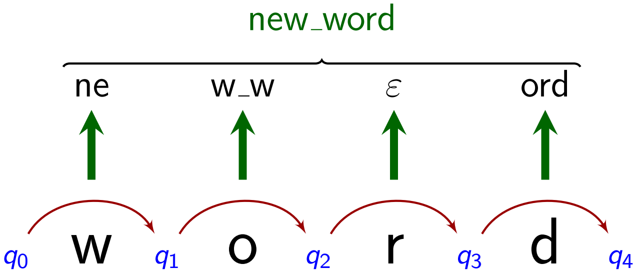
Finite State Automata – On Strings



Transducer

$$q, s \quad \rightarrow \quad q', w$$

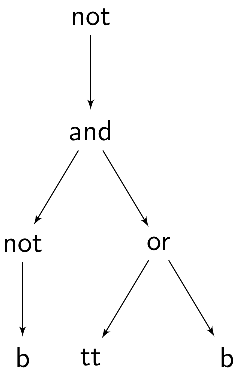
Finite State Automata – On Strings



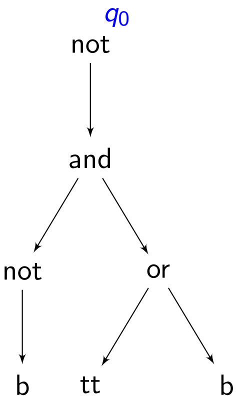
Transducer

$$q, s \quad \rightarrow \quad q', w$$

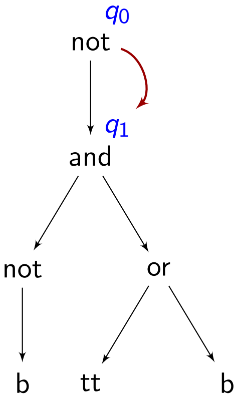
Now on Trees!



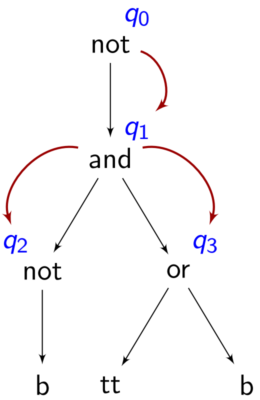
Now on Trees!



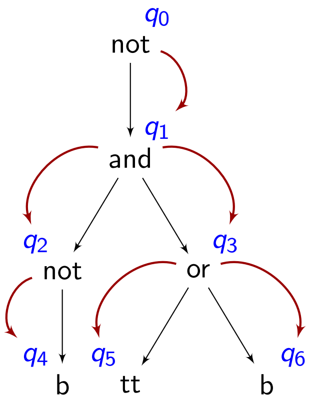
Now on Trees!



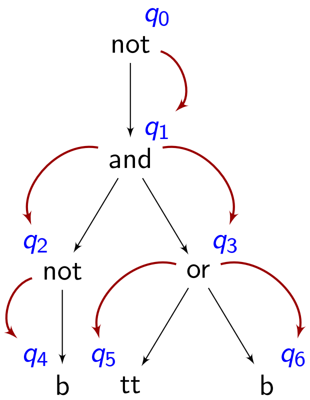
Now on Trees!



Now on Trees!

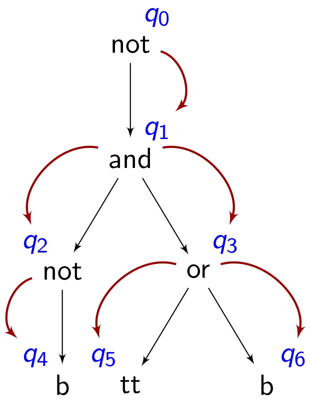


Now on Trees!



$$q, f \rightarrow q_1, \ldots, q_n$$

Now on Trees!



$$q, f \rightarrow q_1, \ldots, q_n$$

Often rendered as a rewrite rule:

$$q(f(x_1, \ldots, x_n)) \rightarrow f(q_1(x_1), \ldots, q_n(x_n))$$

Tree Transducers

$$q(f(x_1, \ldots, x_n)) \quad \rightarrow \quad f(q_1(x_1), \ldots, q_n(x_n))$$

Tree Transducers

$$q(f(x_1, \ldots, x_n)) \quad \rightarrow \quad f(q_1(x_1), \ldots, q_n(x_n))$$

Tree Transducers

$$q(f(x_1, \ldots, x_n)) \quad \rightarrow \quad t[q'(x_i) | q' \in Q, 1 \leq i \leq n]$$

Example: Negation Normal Form

The setting

- Signature: $\mathcal{F} = \{\mathsf{or}/2, \mathsf{and}/2, \mathsf{not}/1, \mathsf{tt}/0, \ \mathsf{ff}/0, \mathsf{b}/0\}$
- State space: $\{q_0, q_1\}$

Example: Negation Normal Form

The setting

- Signature: $\mathcal{F} = \{\text{or}/2, \text{and}/2, \text{not}/1, \text{tt}/0, \text{ ff}/0, \text{b}/0\}$
- State space: $\{q_0, q_1\}$

Transduction rules

$$q_0(\text{not}(x_1)) \to q_1(x_1)$$
$$q_1(\text{not}(x_1)) \to q_0(x_1)$$

Example: Negation Normal Form

The setting

- Signature: $\mathcal{F} = \{\text{or}/2, \text{and}/2, \text{not}/1, \text{tt}/0, \text{ff}/0, \text{b}/0\}$
- State space: $\{q_0, q_1\}$

Transduction rules

$$
\begin{aligned}
q_0(\text{not}(x_1)) &\to q_1(x_1) \\
q_1(\text{not}(x_1)) &\to q_0(x_1) \\
q_0(\text{and}(x_1, x_2)) &\to \text{and}(q_0(x_1), q_0(x_2)) \\
q_1(\text{and}(x_1, x_2)) &\to \text{or}(q_1(x_1), q_1(x_2)) \\
q_0(\text{or}(x_1, x_2)) &\to \text{or}(q_0(x_1), q_0(x_2)) \\
q_1(\text{or}(x_1, x_2)) &\to \text{and}(q_1(x_1), q_1(x_2))
\end{aligned}
$$

Example: Negation Normal Form

The setting

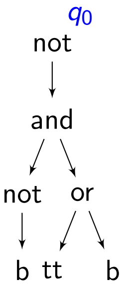
- Signature: $\mathcal{F} = \{\mathsf{or}/2, \mathsf{and}/2, \mathsf{not}/1, \mathsf{tt}/0, \mathsf{ff}/0, \mathsf{b}/0\}$
- State space: $\{q_0, q_1\}$

Transduction rules

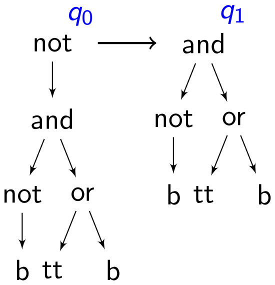
$$
\begin{aligned}
q_0(\mathsf{not}(x_1)) &\to q_1(x_1) \\
q_1(\mathsf{not}(x_1)) &\to q_0(x_1) \\
q_0(\mathsf{and}(x_1, x_2)) &\to \mathsf{and}(q_0(x_1), q_0(x_2)) \\
q_1(\mathsf{and}(x_1, x_2)) &\to \mathsf{or}(q_1(x_1), q_1(x_2)) \\
q_0(\mathsf{or}(x_1, x_2)) &\to \mathsf{or}(q_0(x_1), q_0(x_2)) \\
q_1(\mathsf{or}(x_1, x_2)) &\to \mathsf{and}(q_1(x_1), q_1(x_2))
\end{aligned}
$$

$$
\begin{aligned}
q_0(\mathsf{b}) &\to \mathsf{b} \\
q_1(\mathsf{b}) &\to \mathsf{not}(\mathsf{b}) \\
q_0(\mathsf{tt}) &\to \mathsf{tt} \\
q_0(\mathsf{ff}) &\to \mathsf{ff} \\
q_1(\mathsf{tt}) &\to \mathsf{ff} \\
q_1(\mathsf{ff}) &\to \mathsf{tt}
\end{aligned}
$$

A Run of the Transducer

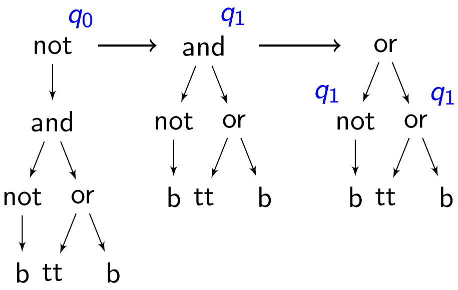


A Run of the Transducer



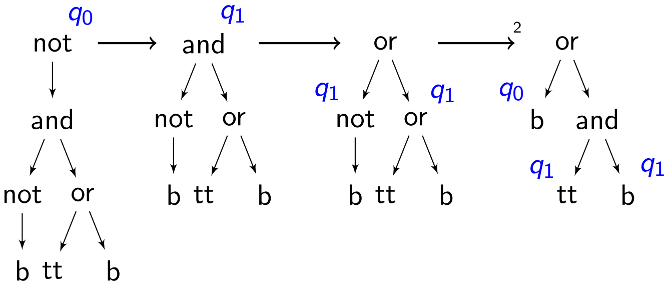
$$q_0(\text{not}(x_1)) \rightarrow q_1(x_1)$$

A Run of the Transducer



$$q_1(\text{and}(x_1, x_2)) \rightarrow \text{or}(q_1(x_1), q_1(x_2))$$

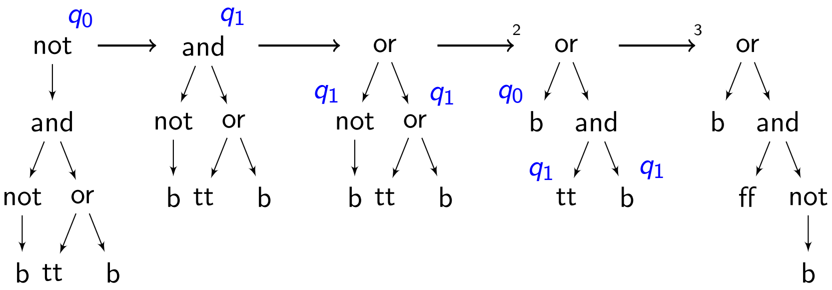
A Run of the Transducer



$$q_1(\mathrm{not}(x_1)) \to q_0(x_1)$$

$$q_1(\mathrm{or}(x_1, x_2)) \to \mathrm{and}(q_1(x_1), q_1(x_2))$$

A Run of the Transducer



$$q_0(\mathsf{b}) \rightarrow \mathsf{b}$$

$$q_1(\mathsf{tt}) \rightarrow \mathsf{ff}$$

$$q_1(\mathsf{b}) \rightarrow \mathsf{not}(\mathsf{b})$$

And now in Haskell

And now in Haskell

Representation in Haskell

type $\mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \rightarrow g^{*}(q, a)$

And now in Haskell

Representation in Haskell

$\textbf{type}\ \mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \rightarrow g^{*}(q, a)$

Free Monad of a Functor g

$$\textbf{data}\ g^{*}\ a = \mathit{Re}\ a \mid \mathit{In}\ (g\ (g^{*}\ a))$$

Example: Substitution

$$\textbf{type}\ \mathit{Var} = \mathit{String}$$

$$\textbf{data}\ \mathit{Sig}\ a = \mathit{Add}\ a\ a \mid \mathit{Val}\ \mathit{Int} \mid \mathit{Let}\ \mathit{Var}\ a\ a \mid \mathit{Var}\ \mathit{Var}$$

Example: Substitution

$$\textbf{type}\ \mathit{Var} = \mathit{String}$$

$$\textbf{data}\ \mathit{Sig}\ a = \mathit{Add}\ a\ a \mid \mathit{Val}\ \mathit{Int} \mid \mathit{Let}\ \mathit{Var}\ a\ a \mid \mathit{Var}\ \mathit{Var}$$

$$
\begin{aligned}
&\mathit{trans}_{\mathsf{subst}} :: \ \mathit{Trans}_{\mathsf{D}}\ \ \mathit{Sig}\ (\mathit{Map}\ \mathit{Var}\ \mu\mathit{Sig})\ \mathit{Sig}\\
&\mathit{trans}_{\mathsf{subst}}\ (m, \mathit{Var}\ v) &&= \textbf{case}\ \mathit{Map.lookup}\ v\ m\ \textbf{of}\\
& && \qquad \mathit{Nothing} \to \mathit{iVar}\ v\\
& && \qquad \mathit{Just}\ t \quad\ \to \mathit{toFree}\ t\\
&\mathit{trans}_{\mathsf{subst}}\ (m, \mathit{Let}\ v\ b\ s) &&= \mathit{iLet}\ v\ (\mathit{Re}\ (m, b))\\
& && \qquad\qquad\ (\mathit{Re}\ (m \setminus v, s))\\
&\mathit{trans}_{\mathsf{subst}}\ (m, \mathit{Val}\ n) &&= \mathit{iVal}\ n\\
&\mathit{trans}_{\mathsf{subst}}\ (m, \mathit{Add}\ x\ y) &&= \mathit{Re}\ (m, x)\ \text{`}\mathit{iAdd}\text{`}\ \mathit{Re}\ (m, y)
\end{aligned}
$$

Example: Substitution

$$\textbf{type}\ \mathit{Var} = \mathit{String}$$

$$\textbf{data}\ \mathit{Sig}\ a = \mathit{Add}\ a\ a \mid \mathit{Val}\ \mathit{Int} \mid \mathit{Let}\ \mathit{Var}\ a\ a \mid \mathit{Var}\ \mathit{Var}$$

$$\textbf{type}\ \mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \rightarrow g^{*}(q, a)$$

$$\begin{aligned}
&\mathit{trans}_{\mathrm{subst}} :: \mathit{Trans}_{\mathrm{D}}\ \mathit{Sig}\ (\mathit{Map}\ \mathit{Var}\ \mu\mathit{Sig})\ \mathit{Sig}\\
&\mathit{trans}_{\mathrm{subst}}\ (m, \mathit{Var}\ v) = \textbf{case}\ \mathit{Map.lookup}\ v\ m\ \textbf{of}\\
&\qquad \mathit{Nothing} \rightarrow \mathit{iVar}\ v\\
&\qquad \mathit{Just}\ t \rightarrow \mathit{toFree}\ t\\
&\mathit{trans}_{\mathrm{subst}}\ (m, \mathit{Let}\ v\ b\ s) = \mathit{iLet}\ v\ (\mathit{Re}\ (m, b))\\
&\qquad (\mathit{Re}\ (m \setminus v, s))\\
&\mathit{trans}_{\mathrm{subst}}\ (m, \mathit{Val}\ n) = \mathit{iVal}\ n\\
&\mathit{trans}_{\mathrm{subst}}\ (m, \mathit{Add}\ x\ y) = \mathit{Re}\ (m, x)\ \text{`}\mathit{iAdd}\text{`}\ \mathit{Re}\ (m, y)
\end{aligned}$$

Example: Substitution

$$\textbf{type}\ Var = String$$

$$\textbf{data}\ Sig\ a = Add\ a\ a \mid Val\ Int \mid Let\ Var\ a\ a \mid Var\ Var$$

$$\textbf{type}\ Trans_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \rightarrow g^{*}(q, a)$$

$$
\begin{aligned}
&trans_{\mathsf{subst}} :: Trans_{\mathrm{D}}\ Sig\ (Map\ Var\ \mu Sig)\ Sig\\
&trans_{\mathsf{subst}}\ (m, Var\ v) = \textbf{case}\ Map.lookup\ v\ m\ \textbf{of}\\
&\qquad Nothing \rightarrow iVar\ v\\
&\qquad Just\ t \rightarrow toFree\ t\\
&trans_{\mathsf{subst}}\ (m, Let\ v\ b\ s) = iLet\ v\ (Re\ (m, b))\\
&\qquad (Re\ (m \setminus v, s))\\
&trans_{\mathsf{subst}}\ (m, Val\ n) = iVal\ n\\
&trans_{\mathsf{subst}}\ (m, Add\ x\ y) = Re\ (m, x)\ `iAdd`\ Re\ (m, y)
\end{aligned}
$$

$$
\begin{aligned}
&subst :: Map\ Var\ \mu Sig \rightarrow \mu Sig \rightarrow \mu Sig\\
&subst = [\![trans_{\mathsf{subst}}]\!]_{\mathrm{D}}
\end{aligned}
$$

Non-Example: Inlining

$$
\begin{aligned}
&\mathit{trans}_{\text{inline}} :: \mathit{Trans}_{\text{D}}\ \mathit{Sig}\ (\mathit{Map}\ \mathit{Var}\ \mu\mathit{Sig})\ \mathit{Sig}\\
&\mathit{trans}_{\text{inline}}\ (m, \mathit{Var}\ v) = \textbf{case}\ \mathit{Map.lookup}\ v\ m\ \textbf{of}\\
&\qquad\qquad \mathit{Nothing} \rightarrow \mathit{iVar}\ v\\
&\qquad\qquad \mathit{Just}\ e \rightarrow \mathit{toFree}\ e\\
&\mathit{trans}_{\text{inline}}\ (m, \mathit{Let}\ v\ b\ s) = \mathit{Re}\ (m\,[v \mapsto b]\,, s)\\
&\mathit{trans}_{\text{inline}}\ (m, \mathit{Val}\ n) = \mathit{iVal}\ n\\
&\mathit{trans}_{\text{inline}}\ (m, \mathit{Add}\ x\ y) = \mathit{Re}\ (m, x)\ \text{`}\mathit{iAdd}\text{`}\ \mathit{Re}\ (m, y)\\
&\mathit{inline} :: \mu\mathit{Sig} \rightarrow \mu\mathit{Sig}\\
&\mathit{inline} = [\![\mathit{trans}_{\text{inline}}]\!]_{\text{D}}\ \emptyset
\end{aligned}
$$

Non-Example: Inlining

$$
\begin{aligned}
&\mathit{trans}_{\mathrm{inline}} :: \mathit{Trans}_{\mathrm{D}}\ \mathit{Sig}\ (\mathit{Map}\ \mathit{Var}\ \mu\mathit{Sig})\ \mathit{Sig}\\
&\mathit{trans}_{\mathrm{inline}}\ (m, \mathit{Var}\ v) = \textbf{case}\ \mathit{Map.lookup}\ v\ m\ \textbf{of}\\
&\qquad \mathit{Nothing} \to \mathit{iVar}\ v\\
&\qquad \mathit{Just}\ e \to \mathit{toFree}\ e\\
&\mathit{trans}_{\mathrm{inline}}\ (m, \mathit{Let}\ v\ b\ s) = \mathit{Re}\ (m\,[v \mapsto b]\,, s)\\
&\mathit{trans}_{\mathrm{inline}}\ (m, \mathit{Val}\ n) = \mathit{iVal}\ n\\
&\mathit{trans}_{\mathrm{inline}}\ (m, \mathit{Add}\ x\ y) = \mathit{Re}\ (m, x)\ \text{`}\mathit{iAdd}\text{`}\ \mathit{Re}\ (m, y)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{inline} :: \mu\mathit{Sig} \to \mu\mathit{Sig}\\
&\mathit{inline} = [\![\mathit{trans}_{\mathrm{inline}}]\!]_{\mathrm{D}}\ \emptyset
\end{aligned}
$$

Recall the type $\mathit{Trans}_{\mathrm{D}}$

$$\textbf{type}\ \mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \to g^*(q, a)$$

Transducers with Polymorphic State Space

The original type $Trans_{\mathrm{D}}$

type $Trans_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \rightarrow g^{*}(q, a)$

Transducers with Polymorphic State Space

The original type $\mathit{Trans}_{\mathrm{D}}$

type $\mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \to g^*(q, a)$

An equivalent representation

type $\mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a . q \to f \qquad a \quad \to g^*(q, a)$

Transducers with Polymorphic State Space

The original type $\mathit{Trans}_{\mathrm{D}}$

type $\mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \to g^*(q, a)$

An equivalent representation

type $\mathit{Trans}_{\mathrm{D}}\ f\ q\ g = \forall a . q \to f(q \to a)\ \to g^*\quad a$

Transducers with Polymorphic State Space

The original type $Trans_{\mathrm{D}}$

type $Trans_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \to g^{*}(q, a)$

An equivalent representation

type $Trans_{\mathrm{D}}\ f\ q\ g = \forall a . q \to f(q \to a) \to g^{*}\quad a$

Deriving the type $Trans_{\mathrm{M}}$

type $Trans_{\mathrm{M}}\ f\ q\ g = \forall\ a . q\ a \to f(q\quad a \to a) \to g^{*}\ a$

Transducers with Polymorphic State Space

The original type $Trans_{\mathrm{D}}$

type $Trans_{\mathrm{D}}\ f\ q\ g = \forall a\,.\,(q, f\,a) \rightarrow g^{*}(q, a)$

An equivalent representation

type $Trans_{\mathrm{D}}\ f\ q\ g = \forall a.q \rightarrow f(q \rightarrow a)\ \rightarrow g^{*}\quad a$

Deriving the type $Trans_{\mathrm{M}}$

type $Trans_{\mathrm{M}}\ f\ q\ g = \forall\ a.q\ a \rightarrow f(q\ (g^{*}\ a) \rightarrow a) \rightarrow g^{*}\ a$

Example: Inlining

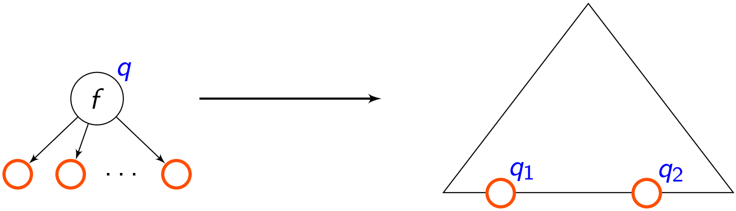
$$
\begin{aligned}
&\mathit{trans}_{\mathsf{inline}} :: \mathit{Trans'}_{\mathsf{M}}\ \mathit{Sig}\ (\mathit{Map}\ \mathit{Var})\ \mathit{Sig}\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Var}\ v) \quad = \textbf{case}\ \mathit{Map.lookup}\ v\ m\ \textbf{of}\\
&\qquad\qquad\qquad\qquad\qquad \mathit{Nothing} \rightarrow \mathit{iVar}\ v\\
&\qquad\qquad\qquad\qquad\qquad \mathit{Just}\ e \quad x \rightarrow e\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Let}\ v\ b\ s) = s\ (m\,[v \mapsto b\ m])\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Val}\ n) \quad = \mathit{iVal}\ n\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Add}\ x\ y) \ = x\ m\ \text{`}\mathit{iAdd}\text{`}\ y\ m
\end{aligned}
$$

Example: Inlining

$$
\begin{aligned}
&\mathit{trans}_{\mathsf{inline}} :: \mathit{Trans'}_{\mathsf{M}}\ \mathit{Sig}\ (\mathit{Map}\ \mathit{Var})\ \mathit{Sig}\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Var}\ v) \quad\ = \mathbf{case}\ \mathit{Map.lookup}\ v\ m\ \mathbf{of}\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{Nothing} \to \mathit{iVar}\ v\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{Just}\ e \quad x \to e\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Let}\ v\ b\ s) = s\ (m\,[v \mapsto b\ m])\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Val}\ n) \quad\ = \mathit{iVal}\ n\\
&\mathit{trans}_{\mathsf{inline}}\ m\ (\mathit{Add}\ x\ y) \ = x\ m\ `\mathit{iAdd}`\ y\ m
\end{aligned}
$$

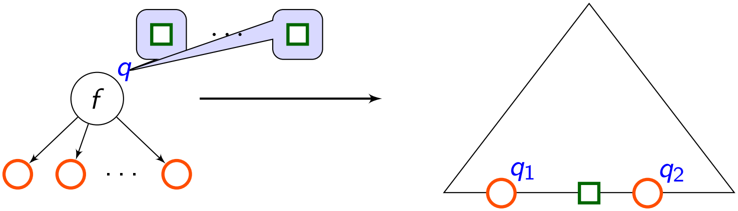
$$
\begin{aligned}
&\mathit{inline} :: \mu\mathit{Sig} \to \mu\mathit{Sig}\\
&\mathit{inline} = [\![\mathit{trans}_{\mathsf{inline}}]\!]_{\mathsf{M}}\ \emptyset
\end{aligned}
$$

Macro Tree Transduction Rule Illustrated



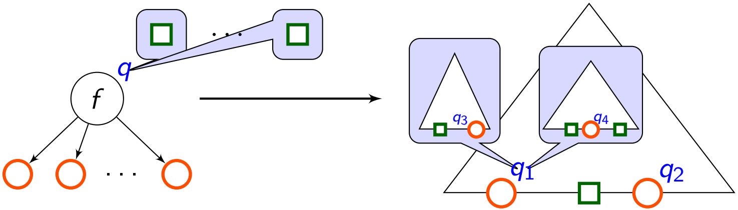
$$\textbf{type } \mathit{Trans}_{\mathrm{M}}\ f\ q\ g = \forall\ a.\ q\ a \to f(\underbrace{q\ (g^{*}\ a) \to a}_{\bigcirc}) \to g^{*}\ a$$

Macro Tree Transduction Rule Illustrated



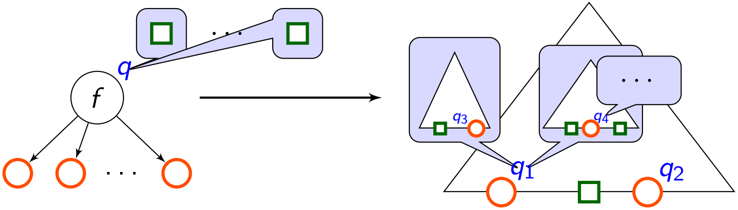
$$\textbf{type}\ \mathit{Trans}_{\mathrm{M}}\ f\ q\ g = \forall\ a.\ q\ a \to f(\underbrace{q\ (g^{*}\ a) \to a}) \to g^{*}\ a$$

Macro Tree Transduction Rule Illustrated



$$\textbf{type}\ \mathit{Trans}_{\mathsf{M}}\ f\ q\ g = \forall\ a.\ q\ a \to f(\underbrace{q\ (g^*\ a) \to a}_{\bigcirc}) \to g^*\ a$$

Macro Tree Transduction Rule Illustrated



$$\textbf{type}\ \mathit{Trans}_{\mathrm{M}}\ f\ q\ g = \forall\ a.\ q\ a \rightarrow f(\underbrace{q\ (g^{*}\ a) \rightarrow a}) \rightarrow g^{*}\ a$$

Conclusion

Why Tree Transducers?

- Compositionality $\rightsquigarrow$ deforestation
- Manipulation of transducers
- Composition with state transition functions

Conclusion

Why Tree Transducers?

- Compositionality $\rightsquigarrow$ deforestation
- Manipulation of transducers
- Composition with state transition functions

Implementation

```
> cabal install compdata
```

Conclusion

Why Tree Transducers?

- Compositionality $\rightsquigarrow$ deforestation
- Manipulation of transducers
- Composition with state transition functions

Implementation

```
> cabal install compdata
```

Future work

- proper formalisation
- monadic transducers
- graph transducers

Bonus Slide: Definition of Macro Tree Transducers

$$q(f(x_1, \ldots, x_n), y_1, \ldots, y_m) \to u$$

for each $f/n \in \mathcal{F}$
and $q/(m+1) \in Q$

Bonus Slide: Definition of Macro Tree Transducers

$$q(f(x_1, \ldots, x_n), y_1, \ldots, y_m) \to u \qquad \text{for each } f/n \in \mathcal{F} \text{ and } q/(m+1) \in Q$$

Where $u \in RHS_{n,m}$, which is defined as follows:

$$\frac{1 \leq i \leq m}{y_i \in RHS_{n,m}} \qquad \frac{g/k \in \mathcal{G} \quad u_1, \ldots, u_k \in RHS_{n,m}}{g(u_1, \ldots, u_k) \in RHS_{n,m}}$$

$$\frac{1 \leq i \leq n \quad q'/(k+1) \in Q \quad u_1, \ldots, u_k \in RHS_{n,m}}{q'(x_i, u_1, \ldots, u_k) \in RHS_{n,m}}$$