Compiling Effectful Terms to Transducers

Prototype Implementation of Memoryful Geometry of Interaction

Koko Muroya
Toshiki Kataoka
Ichiro Hasuo
(Dept. CS, Univ. Tokyo)

Naohiko Hoshino
(RIMS, Kyoto Univ.)
Compiling Effectful Terms to Transducers

Prototype Implementation of Memoryful Geometry of Interaction

Koko Muroya
Toshiki Kataoka
Ichiro Hasuo
(Dept. CS, Univ. Tokyo)

Naohiko Hoshino
(RIMS, Kyoto Univ.)

LOLA (Vienna), July 13, 2014
Consider the term 

\[(\lambda x : \text{nat}. x + x)(3 \sqcup 5) : \text{nat}\]

Our Tool \textit{TtT}

“Terms to Transducers”

\textit{TtT Compiler}

\textit{TtT Simulator}

\textbf{terms}

\textbf{transducers}
Overview

- terms

\( \rightarrow \) \( \lambda \)-terms with algebraic effects

- transducers

\( \rightarrow \) memoryful GoI

[Hoshino, —, Hasuo CSL-LICS ’14]

- simulation result

\( \rightarrow \) stream transducers

- TtT Compiler

- TtT Simulator
Geometry of Interaction (GoI)

- semantics of linear logic proof [Girard ’89],
  functional programming
- token machine presentation [Mackie ’95]
  compilation techniques and implementations
  [Mackie ’95] [Pinto ’01] [Ghica ’07]
Geometry of Interaction (GoI)

- token machine presentation [Mackie '95]

```
0
succ

0
succ

cut
```
Geometry of Interaction (GoI)

- token machine presentation [Mackie ’95]
Geometry of Interaction (GoI)

- token machine presentation [Mackie '95]

proof net style

string diagram style in traced monoidal category
Geometry of Interaction (GoI)

- token machine presentation [Mackie ’95]

proof net style

string diagram style in traced monoidal category
Geometry of Interaction (GoI)

- token machine presentation [Mackie ’95]

proof net style

string diagram style
in traced monoidal category
Geometry of Interaction (GoI)

- token machine presentation [Mackie ’95]

proof net style

string diagram style
in traced monoidal category
Geometry of Interaction (GoI)

- token machine presentation [Mackie ’95]

proof net style

string diagram style in traced monoidal category
GoI is “memoryless”

• advantage: simplicity

• challenges

  • additive connectives \( \&, \oplus \)

  • computational effects
Gol is “memoryless”

- advantage: simplicity
- challenges
  - additive connectives $\&$, $\oplus$
  - computational effects

Laurent ’01
Gol is “memoryless”

- challenge: computational effects

\[(\lambda x : \text{nat}. \ x + x) \ (3 \#\# 5) : \text{nat}\]
GoI is “memoryless”

- challenge: computational effects

$$(\lambda x : \text{nat}. \ x + x) (3 \sqcup 5) : \text{nat}$$
GoI is “memoryless”

- challenge: computational effects

$$((\lambda x : \text{nat}. \; x + x) \; (3 \boxplus 5)) : \text{nat}$$

Diagram:

```
\[\lambda x . x + x \quad 3 \boxplus 5\]
```

```
\text{ask (left) } x
```

Table:

<table>
<thead>
<tr>
<th>[\lambda x . x + x]</th>
<th>[3 \boxplus 5]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ask (left) x</td>
</tr>
</tbody>
</table>
Gol is “memoryless”

- challenge: computational effects

\[(\lambda x: \text{nat}. \ x + x)(3 \sqcup 5) : \text{nat}\]
Gol is “memoryless”

- challenge: computational effects

\[(\lambda x : \text{nat}. x + x)(3 \sqcup 5) : \text{nat}\]
GoI is “memoryless”

- challenge: computational effects

\[(\lambda x : \text{nat}. \; x + x) (3 \sqcup 5) : \text{nat}\]
Gol is “memoryless”

- challenge: computational effects

\[(\lambda x : \text{nat}. x + x)(3 \sqcup 5) : \text{nat}\]
Gol is “memoryless”

- challenge: computational effects

$$(\lambda x : \text{nat}. x + x)(3 \sqcup 5) : \text{nat}$$
Gol is “memoryless”

- challenge: computational effects

\((\lambda x : \text{nat}. x + x)(3 \sqcup 5) : \text{nat}\)
GoI is “memoryless”

- challenge: computational effects

\[(\lambda x : \text{nat}. \ x + x) (3)\]

idea: equip each node with “memory”

<table>
<thead>
<tr>
<th>(\lambda x. x + x)</th>
<th>3 (\sqcup) 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ask (left) (x)</td>
<td>answer 3</td>
</tr>
<tr>
<td>ask (right) (x)</td>
<td>answer 5</td>
</tr>
<tr>
<td>answer (x)</td>
<td>answer 8</td>
</tr>
</tbody>
</table>

This is notable that the underlying proof net—the “graph” on which the current position of the token; a token machines—can itself be a set of algebraic operations, and similarly coproduct types like \(!\text{nat} \times \text{nat}\) or \(\text{nat} + \text{nat}\). The word “memory” here is almost synonymous with “internal states”; we remember the choice by means of its internal state. We use such an idea: equip each node with “memory” in (1).

Here the machine can initially respond to a query \(\text{ask (left)} \ x\) with \(\text{answer} \ 3\), and similarly \(\text{answer} \ 8\) when the choice that it has made, and to \(3\ \sqcup\ 5\) (as a stream) transducer. An example of the limitations of “memoryless GoI” mentioned: computational effects can answer differently. Here what \(\text{ask (left)} \ x\) \(\text{answer} \ 3\) and \(\text{ask (right)} \ x\) \(\text{answer} \ 5\) return.

...compositional manner by means of coalgebraic component calculi.

...with algebraic effects to transducers. Those transducers can be composed resumptions.

...formulated in coalgebraic terms—in transducer translation is based on denotational semantics—given a transducer as a set of algebraic operations, \(\mathcal{B}\), such a set \(\mathcal{D}\) for composing transducers, formulated in coalgebraic terms—in behavioral equivalence. This in fact is already done in [2]: our technique from programs to state machines—“GoI implementation technique from programs to state machines” [Hoshino, —, Hasuo CSL-LICS ’14]...
Memoryful GoI — Input

\[ \lambda \text{-terms with algebraic effects} \]

- algebraic operations [Plotkin, Power ’03]
  - nondeterministic choice
  - probabilistic choice
  - action on global state
Memoryful GoI — Output

stream transducers (Mealy machines)

\[ C = (X, X \times A \xrightarrow{c} T(X \times B), x_0 \in X) \]

automaton style

string diagram style

\[ (T = \mathcal{P}) \]
stream transducers (Mealy machines)

\[ C = (X, X \times A \xrightarrow{c} T(X \times B), x_0 \in X) \]

**automaton style**

\[ (T = P) \]

**string diagram style**
stream transducers (Mealy machines)

\[ \mathcal{C} = (X, X \times A \xrightarrow{c} T(X \times B), x_0 \in X) \]

T = \mathcal{P} \quad (x_0, a_0) \mapsto \{(x_1, b_1), (x_2, b_2)\}

T = \mathcal{D} \quad (x_0, a_0) \mapsto \left[ \begin{array}{c}
(x_1, b_1) \mapsto 1/4, \\
(x_2, b_2) \mapsto 3/4, 
\end{array} \right]
Memoryful GoI — Translation

- idea: resumptions + categorical GoI
  [Abramsky, Haghverdi, Scott ’02]
- use **coalgebraic component calculus**
  [Barbosa ’03] [Hasuo, Jacobs ’11]
  - composition operations for software components
  - (many-sorted) process calculus
Memoryful GoI — Translation

1. introduce component calculus over transducers

2. define interpretation inductively \( (\Gamma \vdash t : \tau) \)

\[
(\Gamma \vdash t \ s : \tau) = (\Gamma \vdash t : \sigma \Rightarrow \tau) \bullet (\Gamma \vdash s : \sigma)
\]

3. prove soundness of interpretation \( (\Gamma \vdash t : \tau) \)
Memoryful GoI — Translation

Def. (component calculus)

\[ C \circ D \]
\[ C \boxplus D \]
\[ \text{Tr}(C) \]
\[ F(C) \]
\[ \overline{\alpha}(\{C_i\}_{i \in I}) \]
**Memoryful GoI — Translation**

**Def. (component calculus)**

\[
(C \circ D) = \left( \begin{array}{c}
Y, \\
Y \times B \xrightarrow{d} T(Y \times C), \\
y_0 \in Y
\end{array} \right) \circ \left( \begin{array}{c}
X, \\
X \times A \xrightarrow{c} T(X \times B), \\
x_0 \in X
\end{array} \right) = \left( \begin{array}{c}
X \times Y, \\
(\ldots, (x_0, y_0) \in X \times Y)
\end{array} \right)
\]

\[
(C \boxdot D) = \left( \begin{array}{c}
X, \\
X \times A \xrightarrow{c} T(X \times B), \\
x_0 \in X
\end{array} \right) \boxdot \left( \begin{array}{c}
Y, \\
Y \times C \xrightarrow{d} T(Y \times D), \\
y_0 \in Y
\end{array} \right) = \left( \begin{array}{c}
X \times Y, \\
(\ldots, (x_0, y_0) \in X \times Y)
\end{array} \right)
\]
Def. (component calculus)

\[
\text{Tr}(C) \quad \equiv \quad \text{F}(C) \\
\overline{\alpha}(\{C_i\}_{i \in I})
\]

(\(\alpha\): \(I\)-ary algebraic operation)
Memoryful GoI — Translation

1. introduce component calculus over transducers

2. define interpretation inductively  \((\Gamma \vdash t : \tau)\)

3. prove soundness of interpretation  \((\Gamma \vdash t : \tau)\)
For a type judgement $\Gamma \vdash t : \tau$ ($\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$), we inductively define

$$
(\Gamma \vdash t : \tau) = \begin{cases}
\frac{}{} & \vdash t : \tau
\end{cases}.
$$
Memoryful GoI — Translation

**Def. (interpretation \((\Gamma \vdash t : \tau)\))**

\[
(\Gamma \vdash t : \sigma) = (\Gamma \vdash t : \tau) \implies (\Gamma \vdash s : \tau)
\]

\[
(\Gamma \vdash \lambda x : \tau. t : \tau \Rightarrow \sigma) = h
\]

\[
A \Rightarrow B \quad \text{intuitionistic logic}
\]

\[
!A \rightarrow B \quad \text{linear logic}
\]
Def. (interpretation \((\Gamma \vdash t : \tau)\))

\[
|\Gamma \vdash n : \text{nat}| = \begin{array}{c}
\hline
h \quad k_n \quad w \quad w' \quad \ldots \\
\hline
\end{array}
\]

\[
|\Gamma, x : \text{nat}, y : \text{nat} \vdash x + y : \text{nat}| = \begin{array}{c}
\hline
h \quad \text{sum} \quad w \quad w' \quad \ldots \\
\hline
\end{array}
\]

\[
|\Gamma \vdash t + s : \text{nat}| = |\Gamma \vdash (\lambda xy : \text{nat}. x + y) t s : \text{nat}|
\]

\[
|\mathbf{x}_1 : \tau_1, \ldots, \mathbf{x}_n : \tau_n \vdash \mathbf{x}_i : \tau_i| = \begin{array}{c}
\hline
\hline
h \quad w \quad w \quad w \quad \ldots \\
\hline
\hline
w' \quad w' \quad w' \quad \ldots \\
\hline
\hline
\end{array}
\]
Memoryful GoI — Translation

1. introduce component calculus over transducers

2. define interpretation inductively $(\Gamma \vdash t : \tau)$

3. prove soundness of interpretation $(\Gamma \vdash t : \tau)$
Memoryful GoI — Translation

Thm. (soundness)

**Theorem 6.2 (Soundness).** For closed terms $t$ and $s$ of type $\tau$,

- If $t \simeq s$, then $([t],[s]) \in \Phi[\tau]$.
- If $t \simeq s$ and $\tau$ is the base type nat, then $\langle t \rangle \simeq_{T_{N,N}} \langle s \rangle$.

where $[\langle t \rangle]$ is the Res($T$)-morphism represented by $\langle t \rangle$, and we write $t \simeq s$ when the equation holds in the extension of the computational lambda calculus. For example, we have

$$\forall (3 \oplus 5) \simeq \forall 3 \oplus \forall 5, \quad 3 \oplus 5 \oplus 3 \simeq 3 \oplus 5 \simeq 5 \oplus 3$$

for any value $\forall$ when the extension of the computational lambda calculus has nondeterminism.
Memoryful GoI — Translation

proof (soundness)

transducers

\[
\begin{array}{c}
B \\
\hline \\
\hline \\
\hline \\
B \\
\end{array}
\]

resumptions

\[
\begin{array}{c}
\hline \\
\cdot \\
\hline \\
\cdot \\
\cdot \\
\hline \\
A
\end{array}
\]

behavioral equivalence
Memoryful GoI — Translation

proof (soundness)

transducers

resumptions

behavioral equivalence

GoI situation \( (\text{Res}(T), \emptyset, \sqcup, \text{Tr}), (F, J_0\phi, J_0\psi, J_0u, J_0v) \)
Memoryful GoI — Translation

proof (soundness)

transducers

resumptions

\( \text{behavioral equivalence} \)

GoI situation

\[ \left( \left( \text{Res}(T), \emptyset, \square, \text{Tr} \right), \left( F, J_0 \phi, J_0 \psi, J_0 u, J_0 v \right) \right) \]

\[ \phi : \mathbb{N} + \mathbb{N} \cong \mathbb{N} : \psi \]

\[ u : \mathbb{N} \times \mathbb{N} \cong \mathbb{N} : v \]
Memoryful GoI — Translation

proof (soundness)

resumptions

partial equivalence relations (per’s) on resumptions

Gol situation

\[ (\text{Res}(T), \emptyset, \Box, \text{Tr}), (F, J_0\phi, J_0\psi, J_0u, J_0v) \]

cartesian closed category \( \text{Per}(T) \)

categorical Gol

[Abramsky, Haghverdi, Scott ’02] realizability
Memoryful GoI — Translation

proof (soundness)

resumptions

partial equivalence relations (per’s) on resumptions

GoI situation

\[ (\text{Res}(T), \emptyset, \Box, \text{Tr}), (F, J_0\phi, J_0\psi, J_0u, J_0v) \]

cartesian closed category \( \text{Per}(T) \) monad \( \Phi \) on \( \text{Per}(T) \)

categorical GoI

[Abramsky, Haghverdi, Scott ’02] realizability
Memoryful GoI — Translation

**proof (soundness)**

<table>
<thead>
<tr>
<th>transducers</th>
<th>resumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>partial equivalence relations (per’s) on resumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartesian closed category $\text{Per}(T)$</td>
</tr>
<tr>
<td>monad $\Phi$ on $\text{Per}(T)$</td>
</tr>
</tbody>
</table>

\[
\llbracket \vdash t : \tau \rrbracket = \text{equivalence class of } \Phi[\tau] \in \text{Per}(T)
\]
## Memoryful GoI — Translation

### Proof (soundness)

<table>
<thead>
<tr>
<th>transducers</th>
<th>resumptions</th>
</tr>
</thead>
</table>

### Denotational semantics

\[
\llbracket \vdash t : \tau \rrbracket = \text{equivalence class of } \Phi[\tau] \in \text{Per}(T)
\]

**Cartesian closed category** $\text{Per}(T)$

**Monad** $\Phi$ on $\text{Per}(T)$

**Partial equivalence relations (per’s)** on resumptions
### Memoryful GoI — Translation

**-proof (soundness)-**

<table>
<thead>
<tr>
<th>transducers</th>
<th>resumptions</th>
<th>partial equivalence relations (per’s) on resumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>[$$\vdash t : \tau$$]</td>
<td></td>
<td>cartesian closed category ( \text{Per}(T) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>monad ( \Phi ) on ( \text{Per}(T) )</td>
</tr>
</tbody>
</table>

\[ [\vdash t : \tau] = \text{equivalence class of } \Phi[\tau] \in \text{Per}(T) \]
Memoryful GoI — Translation

**proof (soundness)**

<table>
<thead>
<tr>
<th>transducers</th>
<th>resumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>partial equivalence relations (per’s) on resumptions</td>
<td></td>
</tr>
<tr>
<td>cartesian closed category $\text{Per}(T)$</td>
<td></td>
</tr>
<tr>
<td>monad $\Phi$ on $\text{Per}(T)$</td>
<td></td>
</tr>
</tbody>
</table>

$$\begin{align*}
(\vdash t : \tau) & \quad \Rightarrow \quad \llbracket \vdash t : \tau \rrbracket = \text{equivalence class of } \Phi[\tau] \in \text{Per}(T)
\end{align*}$$

**denotational semantics**
Memoryful GoI — Summary

use coalgebraic component calculus

(\lambda x : \text{nat}. x + x)(3 \sqcup 5) : \text{nat}
Our Tool \textit{TtT}

\begin{itemize}
  \item \textbf{terms}
  \item memoryful GoI
  \item \textbf{transducers}
  \item \textbf{\(\lambda\)-terms with effects}
  \item \textbf{TtT Compiler}
  \item Haskell program
    \begin{verbatim}
    type Td m x a b = (x, a) -> m (x, b)
    \end{verbatim}
  \item \textbf{TtT Simulator}
  \item simulation result
\end{itemize}
Our Tool \textit{TtT} — Demonstration

\begin{align*}
\text{threeOrFive} &= \text{Oplus \ (Const 3) \ (Const 5)} \\
\text{idOne} &= \text{Apply} \\
&\quad \left( \text{Abst } "x" \ \text{Variable } "x" \right) \\
&\quad \left( \text{Const 1} \right) \\
\text{secondNondetExample} &= \text{Apply} \\
&\quad \left( \text{Abst } "f" \ \text{sumLambda} \right) \\
&\quad \left( \text{Apply \ (Variable } "f" \) \ (\text{Const 0}) \right) \\
&\quad \left( \text{Apply \ (Variable } "f" \) \ (\text{Const 1}) \right) \\
&\quad \left( \text{Abst } "x" \ \text{Oplus \ (Const 3) \ (Const 5)} \right)
\end{align*}
Our Tool $TtT$ — Demonstration

3 □ 5

$(\lambda x. x) \, 1$

$(\lambda f. f \, 0 + f \, 1) \, (\lambda x. 3 \, □ 5)$

(4,526 lines)
Our Tool $TtT$
Our Tool *TtT*

- currently no practical use
Our Tool \textit{TtT}

- currently no practical use
- nevertheless worthwhile
  - helpful for studying higher-order effectful computations
    - showing dynamics of token
  - (speculative) basis of compiler for effectful computations
    - following [Mackie ’95] [Pinto ’01] [Ghica ’07]
Our Tool \textit{TtT}

- currently no practical use
- nevertheless worthwhile
  - helpful for studying higher-order effectful computations
    - showing dynamics of token
  - (speculative) basis of compiler for effectful computations
    - following [Mackie ’95] [Pinto ’01] [Ghica ’07]
- fun to see GoI at work!