From Hilbert space to Dilbert space: Context semantics as a language for games and flow analysis

Harry Mairson
Brandeis University

In memory of my father, Theodore Mairson (1919-2002)
Dear Harry,

I'm very sorry to tell you that your submission #236
How Light Is Safe Recursion? Compositional Translations between Languages of Polynomial Time
to ICFP'03 was rejected.

It was a strong field -- there were 94 submissions altogether, of which we could accept no more than 24. Above and beyond the numbers, the members of the program committee all remarked upon the consistently high quality of the submissions this year; we were forced to reject many fine, publishable papers.

You will find attached reviews of your paper; you can also find them at URL http://edas.cc.gatech.edu/cgi-bin/PaperShow.cgi?m=236 The program committee and their colleagues lavished a huge amount of time and care on these reviews, so I hope that you will find that they offer constructive ideas for how to improve your paper.

Thank you for submitting to ICFP; I am sorry to have to disappoint you.

Best wishes,

Olin Shivers
Program Chair

===== Review =====

*** Rating (10=highest; 1=lowest): 8

*** Confidence (7=high confidence; 1=low confidence): 3

*** Summary (summary of paper): Neergaard and Mairson investigate the construction of a compositional translation of Bellantoni-Cook's functional combinator language BC into Girard-Asperti's Light Affine Logic LAL. Both these languages capture polynomial-time functions.

The paper claims three new results: Murawski and Ong have devised a compositional translation of the linear fragment BC- into LAL. First Neergaard and Mairson show that this fragment can be evaluated in logspace. Second they show that the method used by Murawski and Ong cannot be extended to a translation of the full BC language into LAL. Third they give the first compositional translation of full BC into LAL. This is a new construction where the target code simulates a version of a SECD-machine.

*** Comments (actual review): The general aim behind the research is to develop a better understanding of the essence of the polynomial time complexity class. It is seen as a step in that direction to explain how programming languages that capture p-time can simulate each other. The paper has a rather technical character. In various places remarks are given on what is revealed in this research about the general principles that lie behind. I think that the analysis of this with advantage could have been highlighted and further discussed.

The logspace evaluation trades space for time by handling recursion in a non-standard way without storing the call stack. The authors say they have an implementation on their homepage, the reviewer couldn't find it. Beckmann and Weiermann have given a rewriting system for BC and have shown that it is only polynomial time when evaluated call by value. Therefore to translate the full BC into LAL, it is necessary to capture the evaluation order within the LAL encoding. This requires other methods than given by Murawski and Ong. A SECD machine can be represented as a LAL proof net. The translation takes a BC program into an equivalent LAL proof net simulating a SECD-like machine.

A few opening remarks about the general idea behind the construction in section 5 would improve readability. I would have appreciated if it had been emphasized and specified in more detail, how the implementation handles the two central issues: normal vs. safe and capturing of evaluation order.

The paper reveals aspects of connections between poly-time languages, and places the work in a broader scientific context. A more clear separation of principles in the actual methods used and the technical details (especially in the last part) would let the contribution stand out more and would make it more accessible to a broader audience.

The paper is rather hard to read and follow. It concerns Bellantini and Cook's language BC, and Girard's light linear logic. While there are
From Hilbert space to Dilbert space: Context semantics as a language for games and flow analysis

Harry Mairson
Brandeis University

In memory of my father, Theodore Mairson (1919-2002)
Do laptop presentations make you feel typeset, justified, filed, stamped, indexed, briefed, or de-briefed?

... sometimes when I read stuff put together with some markup language, I feel like a typographical Prisoner ...
Would you rather be typeset, justified, filed, stamped, indexed, briefed, or de-briefed?

Colored pens are friendly, warmer, easier to read (if you write neatly!), more tactile, human, and smell better.

“I am not a number, I am a free man!”
Second Letter.

Fellow Weavers!

Since the time I was a boy, I have been making mats, and I know that all who live in the town of Mecklenburg are familiar with my work. I have made many of these mats, and I always try to make them as fine as possible. I have been working on these mats for over a year now, and I am still working on them. I am writing this letter to the people of Mecklenburg, to let them know what I have been doing.

I have always wanted to do this...
What is context semantics?
[Gonthier, Abadi, Lévy — early 1990s]
Derived from Girard’s geometry of interaction, only without any math, thus “from Hilbert to Delbert.”

Hilbert spaces, C*-algebras, linear operators, execution formula, ... \(\rightarrow\) graphs, stacks, pairs, transitive closure, basic discrete math, ...

Semantics of ... PCF, \(\lambda\)-calculus, fragments of linear logic, ...
Context semantics is...

- a semantics of programs (PCF) and proofs (LL, MELL).
- an exact form of flow analysis, computed statically
  “Does call site X even call function φ?”
- an interactive way of talking about Böhm trees
  and normal forms.
- an explicit mechanization of the kinds of interactions
  found in game semantics [AJM, HON, Mc].
- a distilled form of Lévy’s labelled λ-calculus
  (also a form of program analysis).
- an exact semantics of optimal reduction and maximal
  sharing, providing a proof of its algorithmic correctness,
  with added insights in the presence of linear types.
Outline

- Linear λ-calculus: where graph reduction and context semantics are easy.
- Sharing: boxes and sharing, CBN coding, global, insular, and local graph reduction.
- Readback: how to construct the normal form of a (λ, PCF)-term from its semantics. Relation to flow analysis, games, types.
- Labels: labelled reduction, paths and flows, labels and contexts.
Linear λ-calculus (variable occurs ≤ 1 time)

\[ V \rightarrow x \mid y \mid z \mid \ldots \]
\[ E \rightarrow V \mid \lambda x. E \mid EE \]

Graph coding

Graph reduction

continuation argument

body parameter

principal part

free variables

wire for \( x \)

(\( \lambda x. E \)) \( F \rightarrow E[F/x] \)
Proposition. $S \Rightarrow^* T$ iff $G_S \Rightarrow^* G_T$.

(The graphs are an equivalent, alternative syntax.)

Now, what about a semantics that this graph reduction can be said to preserve?
A graph has parts at its root $p$, and at each of its free variables $\Pi_i$.

**Context Semantics of a Linear $\lambda$-Term:**

$C_E = \{(c, \Pi), (c', \Pi')\} |$  

Context $c$ enters at part $\Pi$, and exits as context $c'$ at part $\Pi'$.

Proposition. If $E \rightarrow^* E'$, then $C_E = C_{E'}$

That's all fine and good—but what does the semantics intuitively tell us about a $\lambda$-term?

[Not only philosophers worry about “The meaning of meaning.”]
Context semantics for linear $\lambda$-calculus

$\Sigma = \{ o, \bullet \}$ tokens $c \in \Sigma^*$ contexts

Contexts are transformed by graph nodes @, $\lambda$:

... semantics preserved by reduction ...
Linear $\lambda$-calculus is strongly normalizing since reduction decreases the graph size.

Context semantics thus describes the normal form—but how?

**EXAMPLE.** $N \equiv \lambda x_0. \lambda x_1. \ldots \lambda x_p. (\ldots (x_i : N_0) N_1) \ldots) N_q$

**INPUT:** $\overset{p+1}{0} \ldots 0$ at the root

**OUTPUT:** $\overset{p+1}{0} \overset{q+1}{\cdot} \overset{q+1}{\cdot}$ at the root

What’s learned? Head variable, number of arguments $x_i$

Now input $\overset{q+1}{0} \cdot \overset{q+1}{0} \cdot \overset{q+1}{\cdot}$ to learn...

Head variable, number of arguments in subterm $N_j$
\[ N \equiv \lambda x_0. \cdots \lambda x_p. \ x_i \ N_0 \cdots (\lambda y_0. \cdots \lambda y_r. \ w \ F_o - F_s) \cdots \ N_q \]

INPUT at root
OUTPUT at root
INPUT at root
OUTPUT at root

\( o^{r+1} \)
\( o^i \circ o^{s+1} \)
\( o^i \circ o^j \circ o^{r+1} \)
\( o^i \circ o^j \circ o^k \circ o^{s+1} \)
\( o^i \circ o^j \circ o^{s+1} \)
\( o^k \circ o^{s+1} \)
\( o^{s+1} \)

"Head variable of \( N \)?"
"\( x_i \)"
"Head variable of \( N_j \)?"
"\( y_k \)"
"\( x_k \)"
"\( w \)" (free)

What is this? Readback: how to read back the normal form of a \( \lambda \)-term from its context semantics

... or equivalently...

A game between an Opponent (computational environment or context) who wants to learn the \( \lambda \)-term known by a Player.
**Readback (cont.)**

\[ E \equiv \lambda x_0. \ldots \lambda x_p. \ x_i \ E_0 \ldots \ E_g \ \text{(assume E closed)} \]

subterms \((E) = \{E\} \cup \bigcup_j \text{subterms} (E_j)\)

headvariables \((E) = \{x_i\} \cup \bigcup_j \text{headvariables} (E_j)\)

Uniqueness of paths: every subterm, head variable has a unique address in \(\{0, 1\}^*\)

\[ \alpha \equiv \bigcirc_{l_1} \ldots a_{l_1} \bigcirc_{l_2} \ldots a_{l_2} \ldots \bigcirc_{l_k} \ldots a_{l_k} \]

parity = \# \ 0 \ in \ context
even \rightarrow \text{subterm/odd} \rightarrow \text{head variable}

Now, find smallest \(k\) where

\((<\alpha 0^k, \text{root}>, <\alpha' 0^k, \text{root}>) \in C_E\)

\(S\) \# \lambda\text{-bindes at top level of } S\)

\# arguments of head variable

What is a "legal" choice of \(\alpha'\)?

What head variables are in view?
Flow analysis: linear terms

\[ \text{not } T = (\lambda p. \lambda x. \lambda y. p y x) (\lambda x. \lambda y. x) \]

\[ \text{Bool} = \alpha \rightarrow \alpha \rightarrow \alpha \]

\[ \text{not: } (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha \]

"Head variable?"

"x" reinterpreted as first argument
Sharing
How do we implement multiple references?
More graph reduction technology...

Boxes

to delineate sharable expressions

croissants to open boxes so their contents can be used...

Call-by-name (CBN) coding

marks a variable occurrence

with sharing nodes to so share them

brackets so one box can absorb another (substitution)

(one among many)

encloses every argument of a function application
CBN coding

Examples

\[ \lambda x. E \]

\[ \text{EF} \]

\[ \lambda z. z \]

\[ \lambda s. \lambda z. s(z) \]
Global reduction: \((\lambda f. f (fy)) (\lambda x. x)\)
Global reduction rules

Global $\beta$-step: Use $\rightarrow$, then other rules as much as possible.
Insular reduction rules: a semantics-preserving alternative

NB: Brackets are detached from boxes
Optimal reduction

- semantics preserving (what semantics?)
- maximal sharing of β-redexes
- local rules only (boxes handled incrementally)

RECALL... that the index of a node is the number of boxes drawn around it

\[ \downarrow \rightarrow \downarrow +1 \]
Context semantics with sharing: the bus

- level of a graph node = \# enclosing boxes
- expand each graph edge to a bus of wires:
context transformations by nodes become level dependent.

\[ \chi_1, \chi_2, \ldots, \chi_k, \chi_{k+1}, \chi_{k+2}, \ldots \]

\[ \chi_1, \chi_2, \ldots, \chi_k, U, \chi_{k+1}, \chi_{k+2}, \ldots \]

\[ \chi_1, \chi_2, \ldots, \chi_k, \chi_{k+1}, \chi_{k+2}, \chi_{k+3}, \ldots \]

\[ \chi_1, \chi_2, \ldots, \chi_k, (\chi_{k+1}, \chi_{k+2}, \chi_{k+3}, \ldots) \]
Sharing node $Y$: change $0, \bullet$ to $L, R$.

**Why All This?**

Refined readback (and flow analysis)

"head variable $x$, $\lambda$-bound at subterm $S$, the $m^{th}$ argument of head variable $y$, $\lambda$-bound at subterm $T$, the $n^{th}$ argument of head variable $z$, $\lambda$-bound at $\ldots$"

$\ldots$ which occurrence of $x, y, z, \ldots$ ?
Readback Revisited

Game: Opponent tries to guess \( \lambda \)-term that is known by the Player

\[ O_{p1}: \quad o^+ \] (initial move)

\[ = \ll o^+, -, -, ..., \gg \]

"What's your head variable?"

\[ P_{l1}: \quad \exists s, o^+ \]

"The left occurrence of \( s \)."

\[ O_{p2}: \quad \exists s, \quad \exists \alpha, o^+ \]

"What's the head variable of its first argument?"

\[ P_{l2}: \quad \exists R<\alpha, s>, o^+ \]

"The right occurrence of \( s \), in the box bound to \( \alpha \)."

\[ O_{p3}: \quad \exists R<\alpha, s>, \quad \exists \beta, o^+ \] "And the head variable of its first argument?"

\[ P_{l3}: \quad \exists \alpha, \exists \beta, \exists z \] "\( z \), in the box bound to \( \beta \), in the box bound to \( \alpha \)."

What Brackets Do:

"internal" sharing of variable in a box

"external" sharing of the box
The context semantics is reversible, and we also have superposition...

- Boolean function evaluation can be superposed (P = NP?)
- Extensions to "higher level sharing"
- Readback: can we read a superposed state?
Why Lamping's optimal evaluation algorithm is correct...

- Readback is correct on graph codings of normal forms — and remains so if the control nodes (4\#, \#) are put in funny places.

- Insular and global reduction produce the same normal forms (up to funny placement of \# \#).

- Insular reduction doesn't change the context semantics used by readback.

- Local reduction (Lamping's algorithm) doesn't change the context semantics at all.
PCF: Adding base types, operations, &c., where linear logic describes sharing.

\[
\begin{align*}
\text{Int} & : \text{Int} \\
\text{Op} & : ? \\
\text{Pl}_1 & : n \\
\end{align*}
\]

\[
\begin{align*}
\text{Int} & : \text{Int} \otimes \text{Int} \\
\text{Succ'} & : \text{Int} \rightarrow \text{Int} \\
\text{Pl}_2 & : o (n+1) \\
\end{align*}
\]

\[
\begin{align*}
\text{Op}_1 & : o ? \\
\text{Pl}_1 & : o ? \\
\text{Op}_2 & : o n \\
\end{align*}
\]

Output ?
Input ?
The input is n
The output is n+1
"Types are arenas for games ('Let's play Int→Int!'), terms are winning Player strategies"
"Thou wall, o wall, o sweet and lovely wall:
Show me thy chink to blink through with mine eyne."

**Geometry of Interaction:**
\( \lambda f. f(f\ 3) \) vs. \( \text{succ} \)

Information flow across two wires

---

\( C_1: \) Left \( f \) asks \( \text{succ} \), "What's the output?"
\( S_1: \) \( \text{succ} \) responds, "What's the input?"

\( C_2: \) Right \( f \) asks \( \text{succ} \), "What's the output?"
\( S_2: \) \( \text{succ} \), "What's the input?"

\( C_3: \) \( R<\alpha, f>, \circ \alpha, 3 \) "The input is 3." (for the second call)
\( S_3: \) \( R<\alpha, f>, 04 \) "The output is 4."

\( C_4: \) \( \text{left} f \), \( \circ \alpha, 4 \) "The input is 4." (for the first call)
\( S_4: \) \( \text{left} f, 05 \) "The output is 5."
Each wire has a dedicated function:

SUCC: \(!\text{Int} \rightarrow \text{Int}\) viewed from the bus

\(\text{Int} \rightarrow \text{Int}^\perp = \text{Int}^\perp\)

\(\text{Int} \rightarrow 0 = 0^\perp = 1_\perp = 1\)
Labelled \( \lambda \)-calculus and paths

\[(\lambda x. E)^l F \rightarrow ([F^l/x]E)^l \] (\(\beta\))

\[(((\lambda x. (x \times 3^4))(\lambda x. (5 \times 6)^7)^8)^9 \]

\[
\uparrow \quad (((\lambda x. (5 \times 6)^7)^{841})(\lambda x. (5 \times 6)^7)^{842})^{349} \\
\uparrow \quad (((\lambda x. (5 \times 6)^7)^{842}8415)(\lambda x. (5 \times 6)^7)^{8428416})^{7841349}
\]

Read \( \cdot \equiv \cdot^R : \quad \cdot \equiv \cdot , \quad \cdot \cdot' \equiv \cdot' \cdot , \quad 8428415 \equiv 8421485 \equiv 215 \]
Sharing! The comonad \((!, \delta, \varepsilon) = (\square, \uparrow A, \uparrow A)\)

\[! : Y \Rightarrow Y\] a covariant functor

\[\delta : ! \Rightarrow \langle\rangle\] natural transformations (polymorphic functions)

\[\varepsilon : ! \Rightarrow 1\]

Unsharing? The monad \((?, \mu, \eta) = (\bigotimes, \downarrow S, \downarrow S)\)

\[?S \leftarrow \mu ?S \Rightarrow ?S \downarrow ?S \leftarrow ?S \downarrow ?S \leftarrow \mu ?S\]

... but what's this got to do with GRAPHS?...
What's all this got to do with \textit{graphs}?

$!\text{(box)}$ is a functor

$!(f \circ g) = !f \circ !g$

(box fusing)

$\delta = \begin{array}{c}
\uparrow \\
\downarrow \\
\uparrow \\
\downarrow \\
\uparrow
\end{array} \\
\varepsilon = \begin{array}{c}
\uparrow \\
\downarrow \\
\uparrow \\
\downarrow \\
\uparrow
\end{array}$

are natural transformations

\begin{align*}
!B & \xrightarrow{\delta_B} !!B \\
!f & \xrightarrow{\circ} !!f \\
!A & \xrightarrow{\delta_A} !!A
\end{align*}

\begin{align*}
!A & \xrightarrow{\epsilon_A} A \\
!f & \xrightarrow{\circ} f \\
!B & \xrightarrow{\epsilon_B} B
\end{align*}
What's all this got to do with \textit{Graphs}?

... comonad equations ...

... look at the bus (context semantics) ...
Conclusions, open problems

- Context semantics: a complete static analysis (as are labelling schemes) — or what can’t it do?
- Where do known flow analyses live within (conservative) or beyond (liberal).
- Slope factor: differentiating variable occurrences (e.g., OCFA).
- Computing intersection types from semantics.
- Full abstraction theorems for sharing, e.g.,
  \[(\lambda x. xx)\ y z \neq y z\ (y z)\].
- Give a context semantics of state, I/O, exceptions, &c. realized by graph reduction and reinterpretation of \((?, \oplus, \downarrow)\).
- Closer relations to games (in a technical sense).
Kienstad terms: context semantics

\[ M_1 = \chi f. f (\lambda x. f(\lambda y. x)) \]
\[ M_2 = \chi f. f (\lambda x. f(\lambda y. y)) \]

\[ \begin{align*}
O_{p_1} &= \circ \\
PL_{p_1} &= \circ Lf \\
O_{p_2} &= \circ Lf, \circ \alpha, \circ \\
PL_{p_2} &= \circ R<\alpha, f> \\
O_{p_3} &= \circ R<\alpha, f>, \circ \beta, \circ \\
(M_1)\ PL_3 &= \circ Lf, \circ \alpha, \circ \beta, \circ y \\
(M_2)\ PL_3 &= \circ R<\alpha, f>, \circ \beta, \circ y 
\end{align*} \]
Kierstead terms: games and readback

\[ M_1 = \lambda f. f_1(\lambda x. f_2(\lambda y. x)) \]
\[ M_2 = \lambda f. f_1(\lambda x. f_2(\lambda y. y)) \]

\[ ((\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}) \rightarrow \text{Int} \]

All moves are \( ? \) ("What's the integer?"")

Player view \( \approx \) lexical scope (compile time environment)

More + pointer chain \( \approx \) context
Conclusions; open problems

- Context semantics is everything - a complete static analysis (as are labelling schemes).
  Where do known flow analyses live within?
  What fragments are tractable?
  (Stop factor: variable occurrences?)

- Context semantics computes Böhm trees (via readback).
  Can it also compute intersection types?

- Context semantics (via cosem, game semantics) is a foundation for full abstraction theorems - what about such a theorem for sharing?

\[(\lambda x. xx)(yz) \neq yz(yz)\]

Applications of y can be shared not here.

- Context semantics is a semantics of comonads
  \((F, \eta, \mu) = (!, \_\_1, \_\_1)\) - why not monads, i.e
  \((F, \eta, \mu) = (?, \_\_2, \_\_2)\) for state, I/O, exceptions, etc, realized via graph reduction.

- Closer relation to games (in technical sense)
Optimal reduction (Lamping)

Local, maximal sharing, semantics preserving

\[ i = j : \]

\[ i \neq j \]

\[ i \leq j = \begin{cases} 1 & \text{if } i < j \\ 0 & \text{otherwise} \end{cases} \]

\[ b = \begin{cases} +1 & \text{if } B = \perp \\ -1 & \text{if } B = \top \end{cases} \]