Towards a Visual Editor for Lens Combinators

Tony Anjorin (Paderborn University, Germany)
Josh Ko (National Institute of Informatics, Japan)

International Workshop on Bidirectional Transformations (BX)
10 April 2018, Nice, France
BiGUL's host language is Haskell
Haskell: 0.231%
My Conjecture

Haskell is a great language with a concise, elegant concrete syntax, but ... 

... it is unfamiliar to most programmers and is thus hard to learn and read

I tried to teach students BiGUL and wound up spending most of the time explaining its cryptic concrete syntax
I tried to teach students BiGUL and wound up spending most of the time explaining its **cryptic** concrete syntax.

Why not establish BiGUL as an external DSL with a truly “natural” concrete syntax?

you can generate whatever you want out of the concrete syntax.
Why not establish BiGUL as an external DSL with a truly “natural” concrete syntax?

So what does “natural” mean?
So what does “natural” mean?

My students (and I tend to agree) say:

VISUAL
So what does “natural” mean?

My students (and I tend to agree) say:

- **VISUAL:**
  - CONNECTORS AND COMPONENTS
  - SIGNALS AND PORTS
  - HIERARCHICAL (ZOOM IN/OUT)

- **Simulink, VHDL**
- **UML, Sysml**
- **component diagrams, ADLs**
- **state charts, Simulink**
Put Block

\[ V \quad b \quad S \]

\[ S \]
Put Block
Put Block

`b.put(s, v)`
Replace

S

Replace

S

S

S
Replace

[Diagram of a process with labeled nodes and connections]
Replace
Skip

\[ V = f(S) \]
Skip

---

S

---

V

---

f

---

S

---

S
Source Rearrangement

\[ \text{RearrS} \]

\[ V \quad b \quad S \quad f \quad S \]
Debugging

replaceHead :=

\[ f = \lambda v \rightarrow (v,()) \]

---

\[ S \rightarrow_{f} \text{RearrV} \rightarrow_{f} \text{RearrS} \rightarrow [S] \]
Debugging

replaceHead :=

\[ f = \lambda v \rightarrow (v,()) \]
Debugging

replaceHead :=

\[
\text{RearrV} \quad f = \lambda v \rightarrow (v,())
\]

\[
\begin{aligned}
42 & \quad S \quad f \quad (42, ()) \\
\text{RearrS} & \quad [S] \\
[S] & \quad []
\end{aligned}
\]
Debugging

\[ f = \langle x:xs \rangle \rightarrow (x, xs) \]

\( (S, ()) \) \( \rightarrow \) \( \text{Prod} \) \( \rightarrow \) \( f \) \( \rightarrow \) \( [S] \)

\( (42, ()) \) \( \rightarrow \) \( f \) \( \rightarrow \) \( [S] \) \( \rightarrow \) \( [] \)
Get Semantics
Get Semantics
Get Semantics

b.get(s)
Skip (Get Semantics)
Skip (Get Semantics)
Skip (Get Semantics)

Switch\( (x, \text{true}, x) \)
Skip (Get Semantics)

\[
\text{Switch}(x, \text{true}, x)
\]
Skip (Get Semantics)

\[ \text{Switch}(x, \text{true}, x) \]
Skip (Get Semantics)

\[
\text{Switch}(x, \text{true}, x)
\]

\[
\text{'}\cdot\text{'}(x, x, x)
\]
Skip (Get Semantics)

\[ \text{Switch}(x, \text{true}, x) \]
\[ \llcorner \cdot \lrcorner(x, x, x) \]
Skip (Get Semantics)

\[='(x, x, \text{true})\]
\[='(x, y, \text{false}) \text{ if } x \neq y\]
Skip (Get Semantics)

`=’(x, x, true)
`=’(x, y, false) if x ≠ y

Switch(x, true, x)

`•’(x, x, x)

S
How This Can Help

• A drag-and-drop visual editor, which is easier to use for programmers not familiar with Haskell

• Novice programmers often need to start from an operational understanding of the language.

• Proficient programmers sometimes also need to debug their program by tracing its execution.

• BiGUL has an axiomatic semantics (to appear in the next session), which currently does not cover lens composition.
Beyond WB Combinators

- An instantiation of the relational/logic programming paradigm (?)
  - Lens combinators are deterministic in both directions.

- Well-behavedness has been regarded as an atomic property established and preserved by lens combinators, but the Skip circuit suggests that there is some “sub-atomic” structure to explore.
  - Prospect for “deterministic relational programming”?
  - Also subsuming reversible programming