Fairness in reactive programming

Andrew Cave

McGill University

January 25, 2013

Joint work with Francisco Ferreira,
Prakash Panangaden and Brigitte Pientka

Introduction

- Reactive programming languages provide a setting to implement GUIs, operating systems, etc.
- Functional reactive programming (FRP) raises level of abstraction [Elliott and Hudak '97]
- Main concepts:
  - *Signals* – time varying values
  - *Signal transformers* – functions from signals to signals
- Unfortunately, many FRP systems allow unimplementable **non-causal** signal transformers

Background

- Linear Temporal Logic (LTL) can act as a type system for (discrete time) FRP (Jeffrey and Jeltsch, 2012). Summarized:

	Logic	Types/Programming
$\Box A$	always A	signals (streams) of As
$\Diamond A$	eventually A	events
$\bigcirc A$	next A	delayed value

- Advantage of logical approach: $\Box A \to \Box B$ captures precisely **causal** stream transformers

Contribution

Our contribution

Convenient ML-like language and type system for writing reactive programs which guarantees **liveness properties**

- The missing piece of the LTL $\leftrightarrow$ FRP correspondence: **proofs** $\leftrightarrow$ **programs**
- Example: A type precisely capturing **causal fair schedulers**

Example 1: Eventualities

- We can write programs like the following:

```
evapp : ◇ A  → □ (A → B) → ◇ B
evapp ea fs = case  ea of
| Now x      ⇒ Now ((head fs) x)
| Later ea’ ⇒ let • ea’’ = ea’
                  • fs’   = tail fs
                in  Later (• (evapp ea’’ fs’))
```

- "Given an eventual A event and a stream of time-varying functions, eventually fire a B event"
- Type **guarantees** eventual delivery of a B (given eventual delivery of the A event)
- Type is a theorem of LTL; program is a proof
- Introduction form for $\bigcirc$ is $\bullet$: delays a value
- Causality enforced: variables bound under $\bullet$ can only be used under $\bullet$ (like Krishnaswami et al, 2011, 2012)

Example 2: Fair schedulers

- A selling point of LTL is its ability to express fairness
- Prior FRP systems cannot express fairness
- We can express a type of fair schedulers:

$$\Box A \to \Box B \to \mathit{Fair}\ A\ B$$

- Any inhabitant is **guaranteed** fair and causal
- What do we mean by "fair"?
  - Infinitely many As and infinitely many Bs

  e.g. the following are considered fair:

  ababababababa...
  babaabaaabaaaab...

Logical foundation for reactive programming

- **Constructive** variant of LTL
- We have only $\bigcirc$ modality
- Adds least and greatest fixed points μ and ν to LTL (in the spirit of the modal μ-calculus)
- The standard modalities are then **defined**:

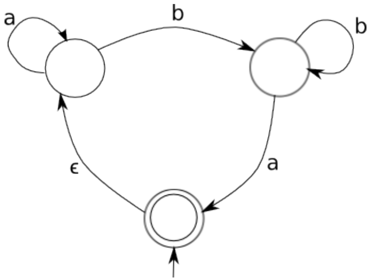
$$\begin{array}{lcl} \Box A & \equiv & \nu X.A \times \bigcirc X \\ \Diamond A & \equiv & \mu X.A + \bigcirc X \\ A \,\mathcal{U}\, B & \equiv & \mu X.B + A \times \bigcirc X \end{array}$$

- Strict positivity restriction on bodies of μ and ν
- We program with **primitive (co)recursion** operators
- i.e. proof terms for ν, μ are (co)induction operators

Expressing fairness in our logical foundation

- Now we can express a type of streams fairly interleaving As and Bs:

$$\mathit{Fair}\ A\ B \equiv \nu X. A\,\mathcal{U}\,(B \times \bigcirc(B\,\mathcal{U}\,(A \times \bigcirc X)))$$



- Can now easily implement a variety of fair schedulers with primitive (co)recursion

Technical Contributions

- A foundational system for programming causal reactive programs with familiar notions of (co)recursion.
  - Suitable for expressing **liveness** properties: **eventualities** and **fairness**
- Type system/proof system is **sound for LTL**
- Operational semantics
- Type safety proofs
- Denotational semantics (in progress)
- Soundness & adequacy proofs (in progress)
- Mechanization (in progress)

Future Work

- Elaboration of structurally recursive programs into our foundational language with primitive (co)recursion operators
  - Structural termination checking and productivity checking
- Prototype implementation (in progress)

Thanks!

In the foundational language with primitive (co)recursion, evapp elaborates to the following:

$$
\begin{array}{l}
\mathit{evapp} : \Diamond A \to \Box(A \to B) \to \Diamond B \\
\mathit{evapp} \equiv \lambda \mathit{ea}.\ \mathsf{prec}\ \mathit{ea}\ (y. \\
\quad \mathsf{case}\ y\ \mathsf{of} \\
\qquad |\mathsf{inl}\ x \mapsto \lambda \mathit{fs}.\ \mathsf{inl}(\ (\mathit{hd}\ \mathit{fs})\ x\ ) \\
\qquad |\mathsf{inr}\ \mathit{frec} \mapsto \lambda \mathit{fs}. \\
\qquad\quad \mathsf{let}\ \bullet \mathit{frec}' = \mathit{frec}\ \mathsf{in} \\
\qquad\quad \mathsf{let}\ \bullet \mathit{fs}' = \mathit{tl}\ \mathit{fs}\ \mathsf{in} \\
\qquad\qquad \mathsf{inr}\,(\bullet\ (\mathit{frec}'\ \mathit{fs}')\,) \\
\quad)
\end{array}
$$

$$\frac{\Gamma \vdash M : \mu X.F \qquad x : F(C) \vdash N : C}{\Gamma \vdash \mathsf{prec}\ M\ (x.N) : C}$$