Towards a Formal Semantics for a Structurally Dynamic Noncausal Modelling Language

John Capper and Henrik Nilsson

Functional Programming Laboratory
Department of Computer Science
University of Nottingham, United Kingdom

TLDI, Philadelphia, January 2012
Outline

1. An overview of equation-based modelling.
2. Functional Hybrid Modelling.
3. A mechanically verified modelling core language.
Describing the **behaviour over time** of physical systems using **sets of equations**.

John Capper and Henrik Nilsson (UoN)
FHM is inspired by Functional Reactive Programming.

However, unlike FRP, the equations in FHM are undirected.

Describing a resistor (Ohm’s Law)

\[ V = IR \]
\[ I = \frac{V}{R} \]
\[ R = \frac{V}{I} \]

More compositional. Promotes reuse of components.

FHM is to logic programming as FRP is to functional programming.
FHM is inspired by Functional Reactive Programming.

However, unlike FRP, the equations in FHM are undirected.

Describing a resistor (Ohm’s Law)

\[ V = IR \]
\[ I = \frac{V}{R} \]
\[ R = \frac{V}{I} \]

- More compositional. Promotes reuse of components.
- FHM is to logic programming as FRP is to functional programming.
Functional Hybrid Modelling

Structurally Dynamic Noncausal Modelling

- FHM is inspired by Functional Reactive Programming.
- However, unlike FRP, the equations in FHM are undirected.

Describing a resistor (Ohm’s Law)

\[ V = IR \]
\[ I = \frac{V}{R} \]
\[ R = \frac{V}{I} \]

- More compositional. Promotes reuse of components.
- FHM is to logic programming as FRP is to functional programming.
FHM is inspired by Functional Reactive Programming.

However, unlike FRP, the equations in FHM are undirected.

Describing a resistor (Ohm’s Law)

\[ V = IR \]

\[ I = \frac{V}{R} \quad R = \frac{V}{I} \]

More compositional. Promotes reuse of components.

FHM is to logic programming as FRP is to functional programming.
The behaviour of a physical system might change dramatically at discrete points in time.

Thus, the equations describing a system may change at runtime. We call this structural dynamism.

We treat equations as first class entities in our language. Thus, new sets of equations may be computed during runtime.
Structurally Dynamic Noncausal Modelling

- The behaviour of a physical system might change dramatically at discrete points in time.
- Thus, the equations describing a system may change at runtime. We call this structural dynamism.
- We treat equations as first class entities in our language. Thus, new sets of equations may be computed during runtime.
The behaviour of a physical system might change dramatically at discrete points in time.

Thus, the equations describing a system may change at runtime. We call this structural dynamism.

We treat equations as first class entities in our language. Thus, new sets of equations may be computed during runtime.
FHM is a functional framework for writing models.

Models are constructed via composition of smaller components.

Models are computed by evaluating an FHM program until an initial set of equations is found.

Events may fire at runtime causing simulation to be suspended while further evaluation is performed to compute a new set of equations. This process is iterated.
Functional Hybrid Modelling

- FHM is a functional framework for writing models.
- Models are constructed via composition of smaller components.
- Models are computed by evaluating an FHM program until an initial set of equations is found.
- Events may fire at runtime causing simulation to be suspended while further evaluation is performed to compute a new set of equations. This process is iterated.
FHM is a functional framework for writing models.

Models are constructed via composition of smaller components.

Models are computed by evaluating an FHM program until an initial set of equations is found.

Events may fire at runtime causing simulation to be suspended while further evaluation is performed to compute a new set of equations. This process is iterated.
Goal: To precisely capture the meaning of the discrete aspects of FHM.

1. (Partial) evaluation of programs to find sets of equations.
2. Responding to events at runtime to generate new sets of equations.
Goal: To precisely capture the meaning of the discrete aspects of FHM.

1. (Partial) evaluation of programs to find sets of equations.
2. Responding to events at runtime to generate new sets of equations.
A Modelling Core Language

The core language is defined via two levels:

1. A time-invariant functional host language.

Syntax

\[
\begin{align*}
  t & ::= x \\
      & | t_1 t_2 \\
      & | \lambda x \cdot t \\
      & | \text{sigrel } z \text{ where } q \\
      & | \ldots \text{ etc} \\
q & ::= s_1 = s_2 \\
    & | t \diamond s \\
    & | q \text{ when } s \\
    & | \ldots \text{ etc} \\
s & ::= z \\
    & | t \\
    & | s_1 + s_2 \\
    & | s_1 \ast s_2 \\
    & | \ldots \text{ etc}
\end{align*}
\]

(don’t worry about the details!)
Step 1: Partially evaluate a modular systems of equations to find an initial flat system of equations. For this, we used Normalisation by Evaluation.

\[
\begin{align*}
\llbracket - \rrbracket : \text{Term} & \to \text{Model} \\
\text{reify} : \text{Model} & \to \text{Nf}
\end{align*}
\]

\[\text{normalisation} \approx \text{reify} \circ \llbracket - \rrbracket\]

Intuitively, normal forms are simply sets of equations (possibly with some additional information).
Task 1: Normalisation (2)

- Reduction free.
- Type directed.
- Can be viewed as a semantic method.
- A constructive algorithm gives rise to a (relatively efficient) implementation.
Task 1: Normalisation (2)

- Reduction free.
- Type directed.
- Can be viewed as a semantic method.
- A constructive algorithm gives rise to a (relatively efficient) implementation.
Task 1: Normalisation (2)

- Reduction free.
- Type directed.
- Can be viewed as a semantic method.
- A constructive algorithm gives rise to a (relatively efficient) implementation.
Task 1: Normalisation (2)

- Reduction free.
- Type directed.
- Can be viewed as a semantic method.
- A constructive algorithm gives rise to a (relatively efficient) implementation.
Iteration

Step II: Responding to runtime events by generating new sets of equations for further simulation. For this, we use Coinduction.

CoLists

```
codata CoList (A : Set) : Set where
  []  : CoList A
  _ :: _  : A \rightarrow \infty CoList A \rightarrow CoList A

run : Term \rightarrow CoList (Set Equation)
run t = ... norm t :: ...
```
Our development has been formalised in Agda and has a number of useful properties:

1. Total and terminating.
2. Productive.
3. Type preserving.
4. Canonical normal forms.
Our development has been formalised in Agda and has a number of useful properties:

1. Total and terminating.
2. Productive.
3. Type preserving.
4. Canonical normal forms.
Properties

Our development has been formalised in Agda and has a number of useful properties:

1. Total and terminating.
2. Productive.
3. Type preserving.
4. Canonical normal forms.
Our development has been formalised in Agda and has a number of useful properties:

1. Total and terminating.
2. Productive.
3. Type preserving.
4. Canonical normal forms.
Correctness

There are also other desirable properties. For example, terms should be convertible ($\beta\eta$-equal) to their normal forms.

\[ \forall t \in \text{Term. } t \equiv_{\beta\eta} \text{norm } t \]
Thank you for listening.
Questions?