More complex datatypes in CellML

Andrew Miller <ak.miller@auckland.ac.nz>
Auckland Bioengineering Institute, University of Auckland

The current situation

- In CellML 1.1 and 1.2, everything is currently treated as a real number.
- There are no functions, vectors, and other constructs which would be useful for many kinds of modelling.
- Without functions, some constructs (such as referring to time-delayed variables) are untidy to implement.

Improving modelling

- A number of typing systems have been considered for CellML 1.2.
- One option would be to make CellML a typed lambda calculus system. In this system, mathematical operations can be performed on types to convert them to other types.
- Types are stored in ordinary variables, just like real numbers are now, and can be imported.

Real numbers

- It makes sense to aggregate the type system with the units system, so that all static checking occurs in one place.
- This suggests built in real numbers like real_metres which encompass both the real number and the type.

Example: complex numbers

- Compute a type from parameters

```
<m:apply
    id="cartesian_complex_type_function_eqn"><m:eq/>
  <m:ci>complex_type_function</m:ci>
  <m:lambda>
    <m:bvar><m:ci>base_type</m:ci></m:bvar>
  </m:lambda>
  <m:apply><c12:vector_type />
    <!-- First argument: type of vector elements...
      -->
    <m:ci>base_type</m:ci>
    <!-- Second argument: size of vector -->
    <m:cn c12:type="real_dimensionless">2</m:cn>
  </m:apply>
</m:apply>
```

i.e.

Functions on types

- This is a function which takes a type, and returns a type, allowing great flexibility when required.
- This great flexibility would allow for parameterised types to be easily created.
- Types have types as well: a function which takes a type and returns a type.

Implementation issues?

- Efficiently implementing the general case of anything more than real numbers is not simple - need complex type inference.
- It has been proposed that we have secondary specifications which cut down what is required to be implemented.
- This would mean that tools can choose the level of complexity they want to support.

Existing solutions

- CellML is not unique in the need to represent different types of data structures.
- Another option would be to make a successor to CellML a subset, or derivative, of an existing language.
- Functional languages like Standard ML and Haskell may already support the features we need, while still being 'pure functional' and so allowing us to keep the procedural details separate from the model.

Discussion

- What capabilities are required for models?
- How general should support be?
- Simple but powerful vs more complex and constrained languages.