Updated proposal for thesis corrections

In order to make the discussion in this document (and in the revised thesis) clear, we distinguish between a *coeffect language framework* and concrete context-aware *domain-specific languages* (DSLs). The former captures the common structure of many context-aware programming languages and is used to simplify the construction of the latter. For example, the coeffect calculus introduced in Section 4.2 of the submitted thesis is a *language framework* while a language for causal data-flow in Section 3.2.4 is a concrete context-aware DSL.

1. Providing uniquely determined semantics

The first issue in the thesis is the incoherent semantics (semantics is defined over a *typing derivation* and different typing derivations lead to a different semantics). We argue that this is a natural property of the *language framework*, which must support multiple different ways in which context is propagated in concrete context-aware DSLs. However, a concrete context-aware DSL should have a coherent semantics. For a given program (written in a given DSL), we intend to choose a *unique typing derivation*. The unique typing derivation then determines the (unique and coherent) semantics of the program.

This will be done formally so that the resulting typing derivation is provably unique. The concrete solution in the revised thesis will be either an algorithm or a syntax driven algorithmic type system. The choice of the typing derivation will be practically justified, which also lets us explore the choices made by existing languages (such as Haskell's implicit parameters) as well as their interesting alternatives.

2. Categorically guided definition of DSLs

The categorical semantics already presented in the thesis (Sections 4.3 and 5.3) will be viewed as a *language framework* for deriving safe *domain-specific languages* with a categorical semantics that guides the implementation of the DSL. We clarify that the categorical semantics is *not* used to prove properties of the system, but merely to guide the design of the concrete DSLs.

The language framework defines a unified type system that can be instantiated to obtain a type system for a concrete context-aware DSL. For a given DSL, the type system prevents certain kinds of errors related to working with context. We will include a formal proof showing what errors are ruled out by the type system for at least 2 concrete coeffect DSLs (discussed in details in Section 4 below).

3. Language framework implementation

As an additional justification for the usefulness of coeffect systems as language frameworks for building safe domain-specific languages, we intend to provide an implementation that transforms code written in a particular DSL (source language with coeffect support) into an existing target programming language. The translation will provide a concrete implementation of the categorical semantics mentioned above (in a similar way in which the Haskell's 'do' notation uses the monadic structure).

The translation will be demonstrated using at least two of the concrete systems discussed in the thesis (implicit parameters, liveness or data-flow) and we will provide a range of concrete example programs that use the domain specific languages (and demonstrate the usefulness of the coeffect abstraction).

4. Proof outline for type soundness of concrete DSLs

The key new results that will be presented in the revised thesis are proofs illuminating the safety guarantees that the coeffect type system of the language framework provides for concrete context-aware DSLs. We intend to provide proof for at least two of the DSLs and give an informal sketch how our proof technique can be generalized to work with any system.

The categorical semantics describes a translation from a source context-aware DSL to a target functional language. We intend to show that for a well-typed program written in the DSL, the translated program can be evaluated in the target language without errors. This will be done by proving the progress and preservation theorems for target language programs that are obtained by the translation.

Using the causal data-flow language as a concrete example, we will:

- Formally define the target language (simply-typed lambda calculus with lists) and its operational semantics (using the standard small-step reduction relation); the language will include exceptions that are produced when a head or tail of an empty list is accessed.
- Define the translation from source DSL into the target language, following the structure of the categorical semantics; the lists in the target language will be used to represent past values; the `prev` construct and variable access will operate on the lists.
- The coeffect type system for data-flow determines the maximal number of past values required by a program. We show that reducing a program obtained from the source DSL with input that has sufficient number of past values does not produce an exception.

The above will be done concretely for two context-aware DSLs and we will also sketch an informal argument showing how the proof would generalize to other DSLs (relying on the abstraction mechanism provided by the type system of Haskell and similar languages).