Motivation: Why context-tracking matters
- Applications today run in diverse environments, such as mobile phones or the cloud. Different environments provide different capabilities, with meta-data and other resources.
- Applications access information and resources of the environment. Such context-dependent interactions are often more important than how the application affects or changes the environment.
- Tracking and verifying how computations affect the environment can be done in a unified way using monadic effect systems, but no such mechanism exists for tracking and verifying how computations access and rely on the context.

Example 1: Liveness analysis & optimization
Annotate variable context with false (0) if it is definitely not live; true (1) if it may be accessed. Unused context can be optimized away.

Context is modelled as dependent Maybe type: $C_1 A = A$ and $C_0 A = 1$.

Example 2: Distributed language with resources
Context carries additional rebindable resources that may be accessed. Annotation specifies a set of resources that are available.

Context is represented using a product type: $C_r A = A \times (r \to \text{Res})$.

Example 3: Efficient data-flow language
Context provides access to previous values of variables. The annotation specifies how many past values may be needed.

Context is represented as a non-empty list; the annotation specifies the length of the list: $C_n A = A \times (A_1 \times \ldots \times A_n)$.

Unified system: Flat coeffect calculus
Captures the essence of context-dependence tracking. Our unified model identifies common properties of the three examples and has desirable theoretical properties (subject reduction and categorical model)
- Sequential composition given by a monoid $(\oplus, \otimes)$ or $(\oplus, \otimes)$
- Context is propagated $(\lor)$ and split $(\land)$ using two additional operators

Generalized system: Structural coeffect calculus
We often need to capture fine-grained structure with context requirements corresponding to individual variables (liveness, data-flow, provenance).
- Compose annotations using a product $(\times)$ that reflect variable structure
- Write system using structural rules that change annotation accordingly