Category-Graded Effect System and Algebraic Theory
Takahiro Sanada
RIMS, Kyoto University, https://www.kurims.kyoto-u.ac.jp/~tsanada/, tsanada@kurims.kyoto-u.ac.jp

Introduction
An important aspect of the theory of programming languages is the treatment of computational effects. Computational effects are the behavior of a program that breaks its feature as a mathematical function, such as input/output, non-deterministic choice, and state reading/writing. Monads and algebraic theories have been studied since 1990s as a notion to treat such computational effects in a unified manner. Recently, category-graded monads are introduced to unify graded monads and parameterised monads. We define category-graded algebraic theory that corresponds to the category-graded monad, and propose an effect system, CatEff, which use it and can safely treat states and input/outputs of varying types.

Examples of Computational Effect and Motivation
Consider the following program with a single state. We assume that the initial state has a value of type Int.

```plaintext
fun p : Int -> Int
let x : Int = read()
let y : Unit = write a
return x
```

The program p reads an integer from the state and bind it to the variable x of type Int, then updates the state with the argument a, and returns the integer x. Next, we consider the following programs:

```plaintext
fun q : Int -> Int
let (x : Int) = read()
if isEven x
then let y : Unit = write true
else let y : Unit = write false
return x
```

The program q is intended to write a boolean value to the state. Unfortunately, this program may cause problematic behavior. Consider calling the program q twice: q (q 42). The first call completes execution successfully. In the second call, x is expected to be an integer, but actually a boolean value. There are several solutions to this problem.

1. Keep the programmer aware of the state of the program and make sure that type inconsistencies do not occur. This solution allows the type of the state to change.
2. Fix the type of state, and detect such a type inconsistency automatically by type checking. This solution frees programmers from using their brains. Programs such as q 42 are no longer accepted.

We propose another solution to this problem by introducing novel notions, called category-graded algebraic theory and category-graded effect. This solution allows the types of the state to change and detects type inconsistencies automatically.

Category
A category C is a directed graph satisfying the following conditions:

- For each vertex A of C, there exists an edge id : A -> A, called identity on A.
- For each pair of edges f : A -> B and g : B -> C, there exists an edge \( g \circ f : A \rightarrow C \), called composition of f and g.
- The following equations hold for any vertices A, B and C:

\[
\begin{align*}
\text{id} & : A \\
\text{id} \circ f & = f \\
(f \circ g) \circ h & = f \circ (g \circ h)
\end{align*}
\]

We write ob(C) for the set of vertex of C, and C(A, B) for the set of edges from A \in ob(C) to B \in ob(C) of C. We call a vertex of a category an object, and an edge a morphism.

An Example of Category-Graded Effect
Let \( S \) be the category “freely generated” by the graph: Int \( \xrightarrow{b} \) Bool. We consider the following read and write operations graded by the morphisms of \( S \).

```plaintext
read : Int \rightarrow \text{read}_\text{Int} \in \text{read}_{\text{Int}}
write : Int \rightarrow \text{write}_\text{Int} : \text{write}_{\text{Int}}
```

Intuition of the graded morphism \( f : A \rightarrow B \) of operations \( \text{op} \) is that \( A \) and \( B \) represent preconditions and postconditions of the effect \( \text{op} \), respectively. For instance, \( \text{write}_a \) can be used under the circumstance that the type of the state is Int, and write boolean value to the state. The programs corresponding to \( p \) and \( q \) in the left column can be written as follows:

```plaintext
fun q' : Int -> Int
let (x : Int) = read()
if isEven x
then let y : Unit = write true
else let y : Unit = write false
return x
```

```
fun q' : Int -> Int
let (x : Int) = read()
if isEven x
then let y : Unit = write true
else let y : Unit = write false
return x
```

<table>
<thead>
<tr>
<th>Program Graded by</th>
<th>The left figure shows some programs and these grading morphisms. ( \odot ) means that the program is not allowed because the programs cause problematic behavior. Such a type inconsistency is detected by type checking, so we can prevent runtime errors. In summary, by grading effect and program, we can prevent runtime errors by type checking while maintaining flexibility of programs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p' )</td>
<td>( \text{id}_{\text{Int}} )</td>
</tr>
<tr>
<td>( q' )</td>
<td>( \text{id}_{\text{Int}} )</td>
</tr>
<tr>
<td>( p' )</td>
<td>( \text{id}_{\text{Int}} )</td>
</tr>
<tr>
<td>( q' )</td>
<td>( \odot )</td>
</tr>
<tr>
<td>( q' )</td>
<td>( \odot )</td>
</tr>
</tbody>
</table>

CatEff [Sanada, 2021]: A Category-Graded Effect System
We give the formal language for category-grade effect and its type system:

**Values:** \( V, W ::= x | \text{true} | \text{false} | n | \ldots \)

where \( x \) and \( n \) ranges over a set of variables and natural numbers, respectively.

**Computations:** \( M, N ::= \text{return}_\alpha V | \text{let} x = M \in N | \text{op} V | \ldots \)

where \( \alpha \) ranges over \( \text{ob}(S) \). We define the rule to derive a typing judgement \( \Gamma \vdash \text{let} x = M \in N : \beta \) when \( \Gamma \vdash \alpha \) and \( \beta \) are type.

```
Gamma \vdash \text{return}_\alpha V : \alpha
Gamma \vdash \text{let} x = M \in N \beta
```

Category-Graded Algebraic Theory
The theoretical background of category-graded effect is category-graded algebraic theory. The right figure is a tree representation of the term \( \text{Term}_G(f, X) \) of \( G \) graded graded algebraic theory. The formal definition of terms of category graded algebraic theory is as follows:

\[
\begin{align*}
d & \in \mathbb{S} \quad \sigma & \in \Sigma \quad \ell & \in \ell \quad \mathbf{n} & \in \mathbf{n} \\
\end{align*}
\]

**Definition** [Sanada, 2021] Let \( X \) be a set, \( S \) be a category, and \( \Sigma \) be \( S \)-graded signature, that is a set of operation symbols. For each operation symbol \( \sigma \in \Sigma \), a morphism \( \ell \) of \( S \) and a natural number \( \mathbf{n} \) are assigned. The set \( \text{Term}_G(f, X) \) of \( G \)-graded \( S \)-term is defined inductively as follows:

\[
\begin{align*}
\mathbf{a} \in \text{ob}(\mathbb{S}) \times X \quad \sigma \in \Sigma \quad \ell_1, \ldots, \ell_n \in \ell \quad \mathbf{n}_1, \ldots, \mathbf{n}_n \in \mathbf{n} \\
\sigma(\ell_1, \ldots, \ell_n) \cdot \ell_1. \ldots \ell_n X & \in \text{Term}_G(f, X)
\end{align*}
\]