Higher Order Operator Overloading for Mathematical Loops

by Sven Nilsen, 2019

Assume the following expression:

$$\sum i \{ x[i] \}$$

One can think about `x` as a function:

$$x : nat \to real$$

Using higher order operator overloading, one can write the sum loop as the following:

$$\sum \{ x \}$$

Omitting the index in the loop means that higher order operator overloading is used inside the body. Next, the implied index of the loop is used as argument to the resulting closure/lambda.

For example:

$$\sum \{ x \cdot y \}$$

$$x : nat \to real$$
$$y : nat \to real$$

This is the same as (applying higher order operator overloading):

$$\sum \{ \backslash(i : nat) = x[i] \cdot y[i] \}$$

Using the implied index:

$$\sum i \{ (\backslash(i : nat) = x[i] \cdot y[i])(i) \}$$

One can see that this gives the same result as:

$$\sum i \{ x[i] \cdot y[i] \} = \sum \{ x \cdot y \}$$

From higher order operator overloading with function currying, it generalizes to packed loops:

$$\sum \{ x \} = \sum i_0, i_1, i_2, \ldots, i_{n-1} \{ x[i_0][i_1][i_2][...][i_{n-1}] \} \quad \text{where} \quad \dim(x) = n$$

This holds for all mathematical loops, such as `∏`, `∀`, `∃`, `min`, `max` etc.