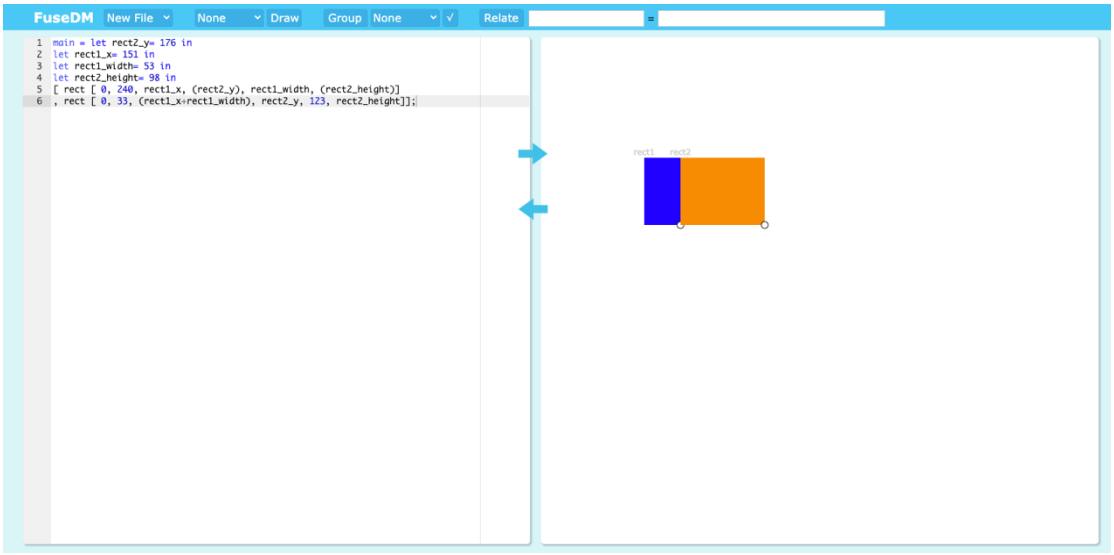
README

Content

Introduction

Our paper (Fusing Direct Manipulations into Functional Programs) proposes a new operation-based framework for bidirectional live programming with a key technique that can fuse direct manipulations into general-purpose functional programs. The artifact FuseDM is a prototype tool to support our operation-based bidirectional live programming framework. As shown below, FuseDM supports developers not only to write functional programs on the left editor and get the output (i.e., SVG) on the right, but also to directly manipulate the output on the right, and automatically synchronize the left code to get the manipulated output. FuseDM offers a series of direct manipulations, as listed in Table 5 of our paper, to edit the output SVG graphics. We successfully designed 14 benchmark examples starting from blank code using direct manipulations supported by FuseDM.



We're applying for the "Reusable" badge. We believe that the artifact deserves that badge because FuseDM can support the main claims of the paper and the code structure is clear and reusable. We recorded two detailed videos (https://youtu.be/QzVbj3IanrI and https://youtu.be/goL_UWXhXKY) to show how direct manipulations supported by FuseDM work and how to reproduce 14 benchmark examples from Sketch-n-Sketch, as mentioned in "Evaluation Instructions". Besides, we explain how to extend FuseDM in two directions, i.e., extending direct manipulations and extending the delta language, in "Addition artifact description".

If reviewers have any problems, please contact us on the artifact submission system and we are always available to resolve all problems.

Claims made by our paper

In Section 5 of our paper, we first demonstrate the expressiveness of the delta language DM that can describe a series of common direct manipulations for editing SVG output in FuseDM. Then we demonstrate that the fusion of direct manipulations in FuseDM into functional programs is effective by successfully implementing 14 benchmark examples from Sketch-n-Sketch.

1. Expressiveness of the Delta Language DM

As mentioned in Section 5.1, using our delta language DM, we implement a series of common direct manipulations in FuseDM for editing SVG output, as shown in the following table.

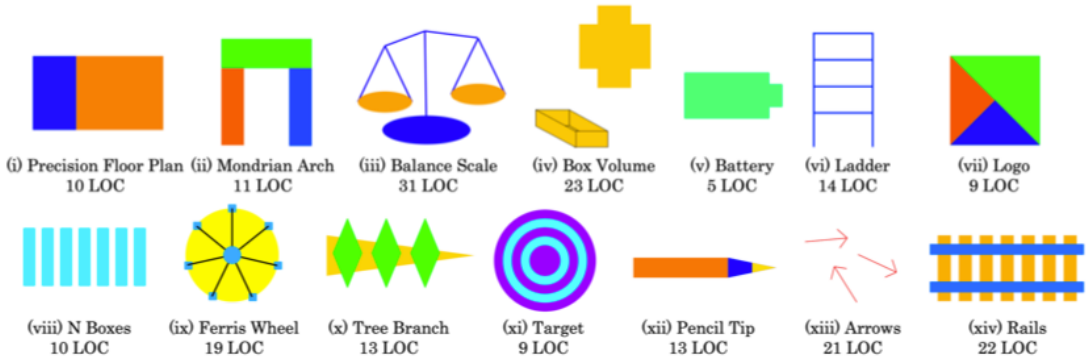
Table 5. Code-insensitive Direct Manipulations in FuseDM

Basic Ops	#Ex	Relating Ops	#Ex	Recursive Ops	#Ex	Mixed Ops	#Ex
Draw Graphic	14	Copy	10	★ Every n	1	Equidistant	4
Drag Graphic	14	Group	8	Drag Group	2	★ Equiangular	1
Resize Graphic	14	Relate	14	Resize Group	1	★ Equiradiidiff	1
Set Color	14					★ Align Center	1
Rotate	1					Align X/Y-axis	2
Delete	0						

The #Ex column indicates the number of examples in Figure 17 in which the operation is used.
Those marked with ★ are extended direct manipulations that are not supported by Sketch-n-Sketch.

2. Effectiveness of Fusion

As mentioned in Section 5.2, using direct manipulations supported by FuseDM, we implement 14 benchmark examples from blank code, as shown in the following figure.



LOC denotes the number of lines of generated source code.

Fig. 17. Benchmark Examples

By implementing those examples, we claim that the strengths of FuseDM are: (1) It allows developers to focus on manipulating outputs instead of source programs. (2) It

guarantees that the manipulated output can be obtained through executing the modified program fused with the direct manipulation.

Download, installation, and sanity-testing instructions

Download

The artifact FuseDM (86.6MB) can be downloaded (in about 5 min) from the Zenodo Link: https://zenodo.org/record/8419914.

Installation

After downloading the FuseDM.zip file, first unzip it into a folder named FuseDM. Reviewers can directly open index.html in the root directory for artifact evaluation, because the compiled file elm.js has been included in index.html. However, extended development needs to install Elm v0.19 and compile the project. Then there are two ways for complete installation of FuseDM: one way is to configure the environment directly on localhost; the other way is to use docker.

1. Install Using Docker (Recommended)

The docker image FuseDM.tar can be found in the root directory of the folder FuseDM.

(1) Download and install docker (https://www.docker.com/).

(2) Import the docker image FuseDM.tar and name it "fusedm" by running

```
docker import FuseDM.tar fusedm:latest
```

(3) Check if import successfully by running

```
docker images
```

Then you will see a new image record as follows.

```
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
fusedm              latest    3e42c907ff44   22 seconds ago   221MB
```

(4) Create a container of the image fusedm with port mapping (port 8000 of the container is mapped to port 8080 of the host.), name it FuseDM, and run the container.

```
docker run -ti --name BiOOP -p 8080:8000 fusedm /bin/bash
```

(5) Enter the project.

```
cd /root/FuseDM
```

(6) Compile the project. This step can be omitted because we have compiled the project in the docker image.

```
elm make src/Main.elm –output=elm.js
```

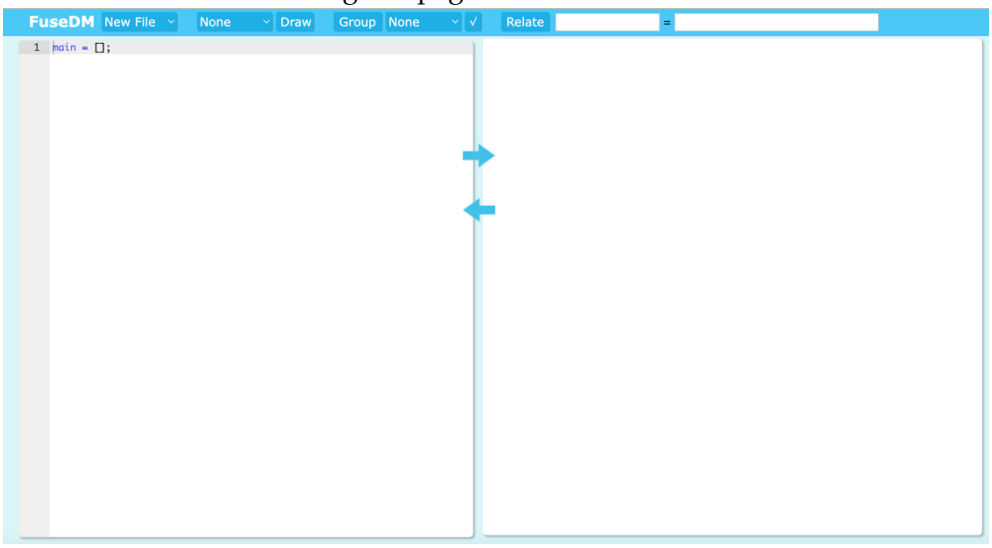
(7) Start a server. (The default URL is http://localhost:8000)

```
elm reactor
```

(8) Open the webpage in the host by visiting the url:

http://localhost:8080/index.html

You will see the following webpage.



2. Install Locally

(1) Install Elm v0.19 (https://guide.elm-lang.org/install/elm.html).

(2) Enter FuseDM folder (The root directory containing the "src" folder).

(3) Compile Project.

```
elm make src/Main.elm --output=elm.js
```

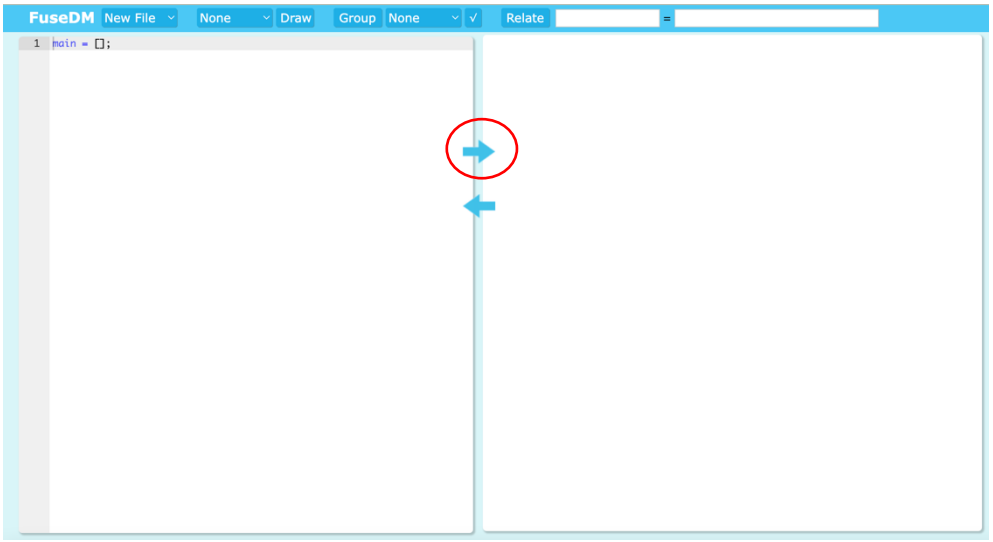
(4) Start a server.

```
elm reactor
```

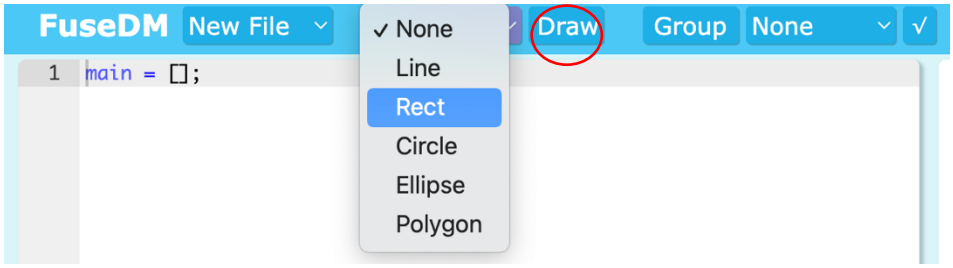
(5) Open http://localhost:8000/index.html using browser.

Sanity-Testing Instructions

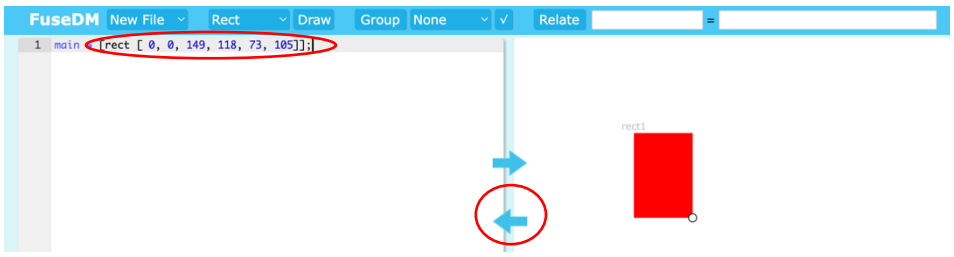
1. Click the blue right arrow and run the left source code "`main = [];`" (generating an empty list). Then you will see nothing on the right.



2. Choose “Rect” and click “Draw” button.



3. Drag and release your mouse on the right output window to draw a red rectangle.
4. Click the blue left arrow. Then the left code changes to “`main = [rect [0,0,…]];`”. This indicates that your installation is successful!



Evaluation instructions

As mentioned in “Claims Made by Our Paper”, there are two claims to be evaluated. The first claim is that FuseDM supports all direct manipulations in Table 5 of our paper. The second one is that we can implement 14 benchmark examples from blank code using direct manipulations supported by FuseDM.

For testing direct manipulations listed in Table 5 of our paper

As shown in Table 5, there are four kinds of direct manipulations: basic operations, relating operations, recursive operations, and mixed operations. We have **recorded a tutorial video on how to use each of the direct manipulations** (the link is https://youtu.be/QzVbj3IanrI). We give detailed steps and believe that reviewers can successfully try them based on our video. Below, we list the point in time when each direct manipulation begins in the video.

- Draw Graphics 00:04
- Drag Graphics 00:13
- Resize Graphics 00:20
- Set Colors 00:26
- Rotate 00:34
- Delete 00:42
- Copy 00:51
- Relate 01:04
- Group 01:17

- Every N 01:32
- Drag Group 01:50
- Resize Group 02:01
- Rotate Group 02:09
- Align X/Y-axis 02:23
- Equidistant 02:41
- Align Center 02:59
- EquiradiiDiff 03:14
- Equiangular 03:30

Note that you should follow the "**(1) run the code; (2) manipulate the output; (3) update the code**" process for each direct manipulation during the testing. Specifically, you should

(1) Click the right-facing arrow to run the source program before each direct manipulation.

(2) Click the left-facing arrow immediately after performing a direct manipulation, such as dragging or resizing a graphic. FuseDM currently supports only single direct manipulation to modify the left program. (It doesn't conflict with our claim.)

Below we provide **source code for some of the direct manipulation testing** to make it easier for reviewers to be able to directly manipulate the existing graphics, since not all direct manipulations can be performed on a blank output to begin with. The code is simple because it is just for testing various direct manipulations.

(1) For relating operations (Copy, Group, and Relate).

```
main = [ rect [0, 0, 141, 122, 57, 108]
, rect [0, 232, 152, 309, 125, 59]];
```

(2) For recursive operations (Every N, Drag Group, and Resize Group).

```
graphics =
g [ 0,
[ rect [0, 184, 82, 116, 18, 91]
, rect [0, 184, 111, 116, 18, 91]
, rect [0, 184, 140, 116, 18, 91]
, rect [0, 184, 169, 116, 18, 91]
, rect [0, 184, 198, 116, 18, 91]
, rect [0, 184, 227, 116, 18, 91]
, rect [0, 184, 256, 116, 18, 91]]];
main = graphics::[];
```

(3) For mixed operations.

a) For Align X/Y-axis and Equidistant operations.

```
graphics =
g [ 0,
    [ rect [0, 184, 48, 114, 18, 91]
    , rect [0, 184, 83, 135, 18, 91]
    , rect [0, 184, 116, 57, 18, 91]
    , rect [0, 184, 142, 126, 18, 91]
```

```
      , rect [0, 184, 198, 78, 18, 91]
      , rect [0, 184, 237, 148, 18, 91]
      , rect [0, 184, 275, 97, 18, 91]]];
main = graphics :: [];
```

b) For Align Center and Equiradiidiff operations.

```
main =
let graphics =
g [ 0,
        [ circle [0, 36, 137, 131, 110]
        , circle [0, 48, 269, 314, 75]
        , circle [0, 13, 307, 134, 56]]] in
graphics :: [];
```

For reproducing 14 benchmark examples shown in Figure 17 of our paper

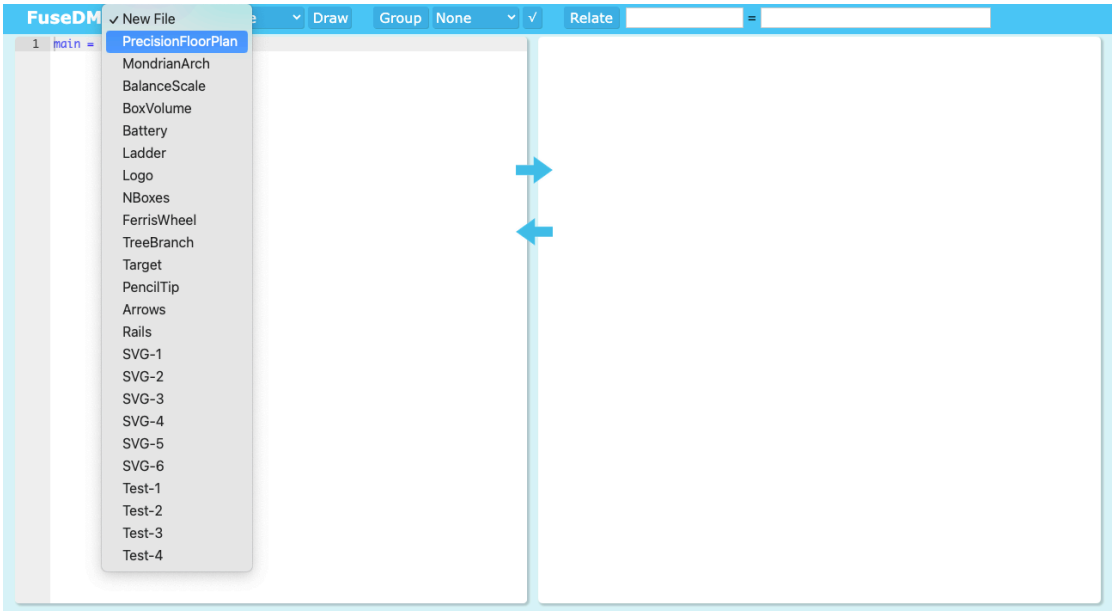
As shown in Figure 17, we reproduce 14 examples from Sketch-n-Sketch using direct manipulations supported by FuseDM. These examples were implemented from blank code, and all are done almost exclusively using direct manipulations on the output without the need to edit the code. Note that there is a special case where the position of the list elements in the code needs to be manually adjusted in order to adjust the overlapping relationship of the graphics on the output, such as examples Box Volume and Ferris Wheel.

We have also **recorded a detailed aggregated video on how to reproduce each of the benchmark examples** (the link is https://youtu.be/goL_UWXhXKY). Reviewers can reproduce the 14 examples following our video. Below, we list the point in time when each example begins in the video.

1. Precision Floor Plan 00:03
2. Mondrian Arch 00:46
3. Balance Scale 01:51
4. Box Volume 03:49
5. Battery 06:30
6. Ladder 07:12
7. Logo 08:02
8. N Boxes 09:25
9. Ferris Wheel 10:09
10. Tree Branch 12:03
11. Target 13:47
12. Pencil Tip 14:25
13. Arrow 17:03
14. Rails 18:08

If you think the video is a bit too fast and unclear, you can find the original video that wasn't accelerated is at link https://drive.google.com/drive/folders/1qrtLjoOl-Z12BAqjVnyMUT0AA11Fdjyh?usp=sharing. In this link, each example is a separate video.

Besides, we put the generated code (on the left window) for each example (after finishing all direct manipulations on the output window) into the source code folder with the path "FuseDM/src/Examples". The generated code is a bit messy, so we organized it manually for easier reading. Reviewers can also run the generated code for each example by choosing the prepared files as follows. (Only if a server is started, i.e., after running the command `elm reactor`, you can select an example code directly from the menu bar. Otherwise, you can only manually copy the code into the editor. In both ways, you can interact with FuseDM.)



Additional Artifact Description

File Structure

- **ace**: The source code of the code editor (left side of the window) that we import in FuseDM.
- **arrow1.png**: The picture for the right-facing arrow.
- **arrow2.png**: The picture for the left-facing arrow.
- **elm-stuff**: It is used to manage project dependencies and store generated files. Typically, you don't need to manually manipulate this folder.
- **elm.js**: The JavaScript file generated by Elm compiler. It is included in the index.html file to make the project functional in a web browser.
- **elm.json**: A configuration file that manages project dependencies and specifies various project settings.
- **FuseDM.tar**: The docker image that contains everything for artifact evaluation and extended development of FuseDM.
- **index.html**: The entrance of the compiled project and can be opened in a browser.
- **main.css**: The file to specify the style of HTML elements.

- **README.md**: Not completed. Later this README for artifact evaluation will replace it.
- **src**
  - **Examples**: The generated code for 14 benchmark examples and other test cases.
  - **Language (The Core)**:
    - **BEval.elm**: The fusion algorithm given in Section 4.2.
    - **FEval.elm**: The forward evaluation given in Figure 11.
    - **Fusion.elm**: Transformation defined in Figure 15 that transforms a delta to an expression function.
    - **Syntax.elm**: Syntax of the delta language DM defined in Section 3.1 and the core functional language F defined in Section 4.1.
    - **Utils.elm**: Implement the important optimized merge operation defined in Definition 4.4.
  - **Main.elm**: The main file to be compiled.
  - **Model.elm**: Initializes the model for FuseDM application.
  - **Native**:
    - **editTrans.js**: Translate events triggered in SVG output (when a direct action is completed) into the corresponding DM (the delta language) code.
    - **other files**: JavaScript code for direct manipulation on the SVG canvas, such as the underlying code that supports drawing as well as dragging a graphic.
  - **Parser**: Parsers for deltas and expressions.
  - **Printer**: Printers for values and expressions.
  - **Utils.elm**: Auxiliary functions.
  - **View.elm**: Define the visual UI using Elm, like HTML.

Extending FuseDM

There are two directions to extend FuseDM: (1) Adding more customized direct manipulations for editing SVG output; (2) Extending the delta language DM to add more features to allow that more program transformations can be realized by manipulating directly on the output in FuseDM.

1. Adding More Direct Manipulations

a) Add user interfaces for new direct manipulations using JavaScript to the Native folder.

i. For example, if you want to add an operation that makes a list of graphics with a gradient of colors, then you should clarify how to operate on the output (e.g., select a group and select the “color gradient” menu item) and what it looks like when it's done (e.g., let the color values of the graphics be equidistant).

b) Add the translation that translates an GUI operation into the code written in

DM (the delta language) in the file "FuseDM/src/Native/editTrans.js". Specifically, you should generate a string of DM code and set the global variable "gEdit" to that string, therefore, after clicking the left-facing arrow, the system parses the value of "gEdit" and fuses the delta into the left functional program. Therefore, we give the grammar of DM.

i. **Identity**: id
ii. **Graphic Deltas**: Graphic.map <list delta>
iii. **Replacements**: rewr <atomic expressions>
iv. **Compositions**: <delta1> . <delta2>
v. **Arithmetic Deltas**: +n | *n
vi. **Tuple Deltas**: (<delta1>, <delta2>)
vii. **List Deltas**: [<delta1>,<delta2>,…] | delete n | insert n <atomic expressions> | modify n <delta>
viii. **List Folding:** {<todelta>|<derive> <| acc}
e.g., {\x->Graphic.map modify 2 (rewr x)| \i->(x1 +i* dis, i+1)<|0}
ix. **Constraint Creations**: ?x=><delta>, where extracting a sub-expression and assigning it to the variable "x" can use "&x". For example, if you want to make the second element of a tuple to be twice as big as the first element, you can write "?x=>(&x, rewr (2*x))"

2. Extending the Delta Language DM

If you want to add some new features, such as abstracting a function and renaming parameters, you can add some special operators (or constructors) to the delta language DM according to the following steps.

a) Add the syntax of the operator (called OP below) for customized features to "FuseDM/src/Language/Syntax.elm"
b) Add the application rule for OP to "FuseDM/src/Language/BEval.elm"
c) Add the transformation that translates OP into an expression function to "FuseDM/src/Language/Fusion.elm"
d) Add the parser for the operator to "FuseDM/src/Parser/Delta.elm"