Interactive Stepper for Expression with Holes

YANJUN CHEN*, University of Michigan, USA

1 INTRODUCTION

Stepper that can display all immediate states of a program is a useful tool for debugging and education. It simplifies expression step by step so that people can track the evaluation procedure easily. Hazel, a programming language environment developed by [citation Cyrus Omar], allows to evaluate incomplete programs. It defines a special variable called hole indicating the place that needs to fill. In this paper, we want to develop an interactive stepper for Hazel. Take the following programs as examples:

In Hazel , the incomplete programs are also allowed for evaluation. There are two examples shown in Figure 1.

In the first example, the expression is already simple enough and it is unnecessary to expand cases since the parameter has not provided. Therefore, stepper need to detect this situation so that it will pause until user requires it to progress. Another example has an expression with two parts. The left part is heavy since it takes about 16 steps to final while user might just want to debug the right part. According to the regular dynamics, there should contain only one way to step for any non-final programs. So, user has to click 16 times to reduce the left one in order to debug the right one. The possible solution is just allowing the multiple evaluation contexts so that user can choose the subexpression they need. This paper develops a stepper with multiple evaluation contexts and pausing environment.

2 CONTEXTUAL DYNAMICS

The syntax of DH-types and DH-expression is shown in Figure 2. We use the simplified definition given in [Omar] combined with number and addition.

The structure of the stepper is shown in Figure 3. The expression is first decomposed into many independent evaluation contexts. Then, user may choose one of them to progress. And the stepper will compose them into the result then. For example, in Figure 1, we have 3 evaluation contexts highlighted as green boxes. And when we click the second one, only the expression in second box is evaluated and composed into the whole expression.

Also, for some unnecessary steps, the stepper can detect it and then pause unless user click it manually. So, during the decompose, we need to detect this situation so that the stepper will not go on again. An example is shown in Figure [].....

We now formally define the instruction transition judgement. We use $d_1 \longrightarrow d_2$ as instruction transition judgement. Figure 4 shows some of the transition judgement.

In Hazel, we have four types of expressions: boxed value, indeterminate, step, and paused. The instruction transition can be implemented to provide the type of the expressions. Therefore, we can easily know whether an expression is final, paused or steppable.

Then, we have the evaluation context. Unlike the regular evaluation context, the decompose function will return a list of contexts. The recursive definition of the decompose function is shown in Figure 5.

In conclusion, we first decompose the expression into many evaluation contexts for user to choose. Then, we run the instruction transition and compose the result to user.

*Research advisor: Cyrus Omar; ACM student member number: ; Category: undergraduate

Author's address: Yanjun Chen, University of Michigan, USA, yanjunc@umich.edu.

```
let fibo : Int → Int =
  λx.{
  case x
  | 0 ⇒ 1
  | 1 ⇒ 1
  | 2 ⇒ 2
  | 3 ⇒ 3
  | 4 ⇒ 5
  | 5 ⇒ 8
  | 6 ⇒ 13
  end
  }
in
fibo 1
```

```
TYPE OF RESULT: Int
  case 1:1
  | 0 ⇒ …
  | 1 ⇒ …
  | 2 ⇒ …
  | 3 ⇒ …
  | 4 ⇒ …
  | 5 ⇒ …
  | 6 ⇒ …
  end
```

Fig. 1. Example of Paused Environment

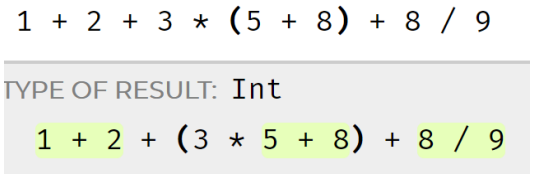

Fig. 2. Example of Multiple Environment

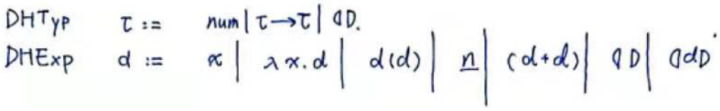

Fig. 3. Syntax

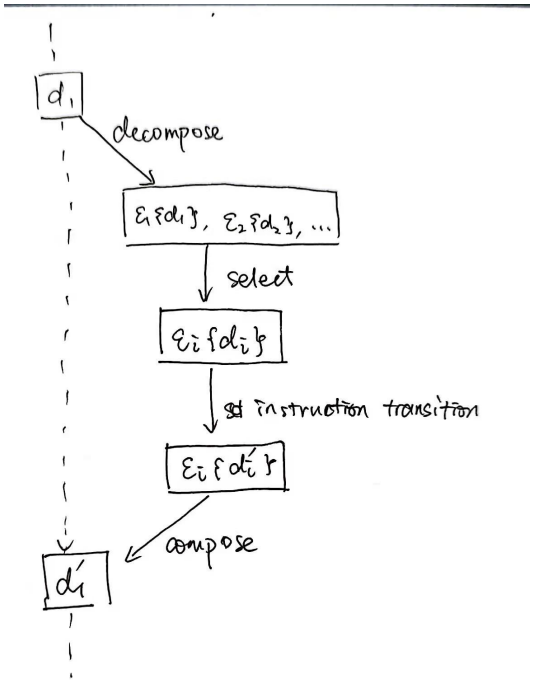

Fig. 4. Sturcture of Stepper

Fig. 5. Instruction Transition

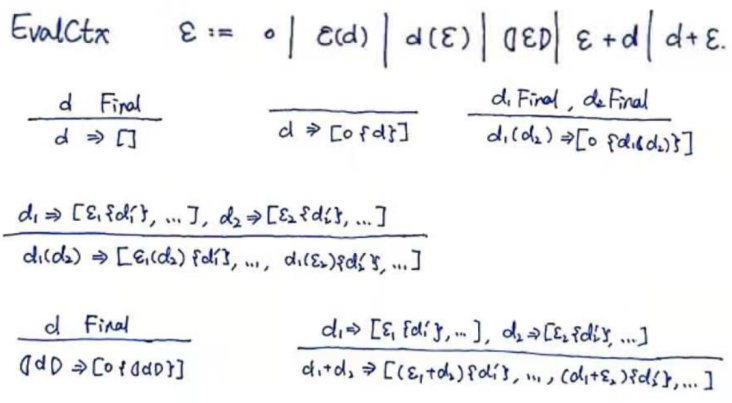

Fig. 6. Decompose