Interactive Stepper for Expression with Holes

YANJUN CHEN*, University of Michigan, USA

1 INTRODUCTION

??

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

(a) Example of paused environment   (b) Example of multiple environment

Fig. 1. Two programs in Hazel

Figure 1a is an example of expression with a hole. The evaluation result is a case expression with empty parameter. However, the origin expression is already simple enough and it is unnecessary to expand like that especially when there are many cases in the function. Therefore, we want to develop a stepper that can detect this situation so that it will pause until user requires it to step further.

Figure 1b shows an expression with three parts. The left part is heavy since it takes many steps to evaluate. However, user might just want to debug the right lambda part. According to the regular dynamics, there should contain only one way to step for any non-final programs. So, user has to click many times to reduce the left one in order to debug the right one. Our solution is to provide options for user to choose where to step. In the Figure 1b, the green boxes contain subexpressions that can be evaluated. So, stepper allows the multiple evaluation contexts so that user can choose the subexpression they need.

Our interactive stepper develops pausing environment and a simple algorithm to decompose multiple evaluation contexts. Furthermore, it also has a webview so that user can click boxes to step expressions.

Contributions. The contributions of this paper are: (1) a pausing environment in section 2; (2) an algorithm to decompose multiple evaluation contexts in section 3.

---

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

---

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

2 PAUSING ENVIRONMENT

We first formally define the simplified syntax of Hazel shown in Figure 2. The symbol $(\!|\!)$ represents the empty hole used to indicate the place that need to fill in the incomplete program. And for the expression that doesn't have clear meaning, we use $(\!|e|\!)$ to indicate that it doesn't have semantic meanings. We only consider binary sums so for $\text{inj}_i(e)$, $i \in \{L, R\}$.

$$HTyp\ \tau\ ::=\ \text{num}\ |\ \tau \to \tau\ |\ (\tau + \tau)\ |\ (\!|\!)^A$$
$$HExp\ e\ ::=\ x\ |\ (\lambda x.e)\ |\ (\lambda x : \tau.e)\ |\ e(e)\ |\ \underline{n}\ |\ (e + e)\ |\ e : \tau\ |\ \text{inj}_i(e)\ |\ \text{case}(e, x.e, y.e)\ |\ (\!|\!)\ |\ (\!|e|\!)$$

Fig. 2. Syntax of H-types, H-expressions, and Type Constraint Set

$\boxed{e\ \text{paused}}$ e is paused

$$\frac{e_0\ \text{indet}}{\text{case}(e_0, x.e_1, y.e_2)\ \text{paused}}$$

Fig. 3. Paused Forms

Normally, we have judgement like final, boxedvalue for a program. Furthermore, since Hazel contains incomplete programs, there exists some indeterminate programs [citation] which induces a judgement called indet. Pausing environment is introduced by an extra judgement, paused. For example, when $(\!|\!)$ appears at case expression so that the match algorithm gives indeterminate result, it will be a paused expression so that the stepper can stop immediately. Figure 4a gives a expression as example. It defines a function func. But we don't provide any argument for it. The evaluator expand the case expression directly. However, Figure 4b shows an example that when augment is only a hole, the stepper will not evaluate but stop with yellow box indicating it is steppable but paused.
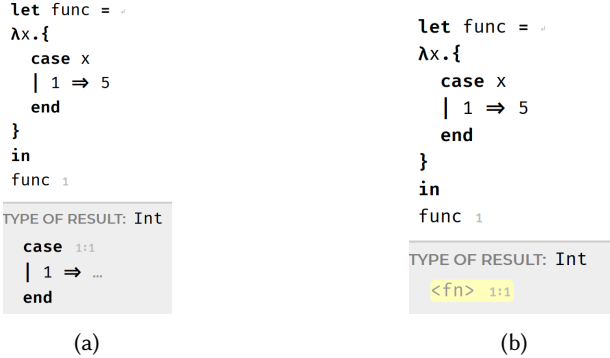
```
let func =
λx.{
  case x
  | 1 ⇒ 5
  end
}
in
func 1

TYPE OF RESULT: Int
  case 1:1
  | 1 ⇒ …
  end
```

```
let func =
λx.{
  case x
  | 1 ⇒ 5
  end
}
in
func 1

TYPE OF RESULT: Int
  <fn> 1:1
```

(a)   (b)

Fig. 4. Example of stepper with paused environment

3 CONTEXTUAL DYNAMICS

The structure of the stepper is shown in Figure 5. 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 1b, 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.

We now formally define the instruction transition judgement. We use $e_1 \mapsto e_2$ as instruction transition judgement. Figure 6 shows some of the transition judgement.
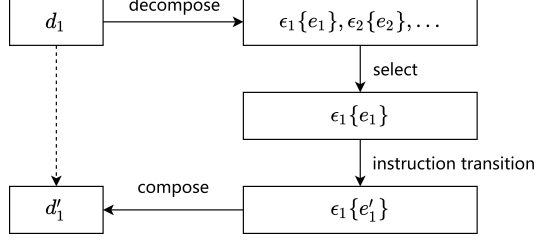
Fig. 5. Structure of Stepper

As what Section ?? discussed, 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 6.

$\boxed{e \mapsto e'}$ e takes an instruction transition to e'

$$\frac{e_2 \text{ final}}{(\lambda x.d_1)(d_2) \mapsto [d_2/x]d_1}$$

$$\text{EvalCtx } \epsilon \quad ::= \quad \circ \mid \epsilon(e) \mid e(\epsilon) \mid (\!(\epsilon)\!) \mid \epsilon + e \mid e + \epsilon$$

$\boxed{e \Rightarrow [\epsilon_1\{e_1\}, \epsilon_2\{e_2\}, \cdots]}$ e is decomposed into contexts $\epsilon_1\{e_1\}, \epsilon_2\{e_2\}, \cdots$

DFinal
$$\frac{e \text{ final}}{e \Rightarrow []}$$

DBoxedVal
$$\frac{e \text{ boxedval}}{e \Rightarrow [\circ\{e\}]}$$

DApFinal
$$\frac{e_1 \text{ final } e_2 \text{ final}}{e_1(e_2) \Rightarrow [\circ\{e_1(e_2)\}]}$$

DAp
$$\frac{e_1 \Rightarrow [\epsilon_1\{e_1'\}, \cdots] \qquad e_2 \Rightarrow [\epsilon_2\{e_2'\}, \cdots]}{e_1(e_2) \Rightarrow [\epsilon_1(e_2)\{e_1'\}, \cdots, e_1(\epsilon_2)\{e_2'\}, \cdots]}$$

DHoleFinal
$$\frac{e \text{ final}}{(\!(e)\!) \Rightarrow [\circ\{(\!(e)\!)\}]}$$

DAdd
$$\frac{e_1 \Rightarrow [\epsilon_1\{e_1'\}, \cdots] \qquad e_2 \Rightarrow [\epsilon_2\{e_2'\}, \cdots]}{(e_1 + e_2) \Rightarrow [(\epsilon_1 + e_2)\{e_1'\}, \cdots, (e_1 + \epsilon_2)\{e_2'\}, \cdots]}$$

Fig. 6. Insturction transition and decomposition

For example, we have an expression $e_0 = 4 + 1 + (5 + 6)$. First, we know that $4 + 1 \Rightarrow [\circ\{4 + 1\}]$ and $5 + 6 \Rightarrow [\circ\{5 + 6\}]$. So, According to DAdd rule, we have,

$$4 + 1 + (5 + 6) \Rightarrow [(\circ + (5 + 6))\{4 + 1\}, (4 + 1 + \circ)\{5 + 6\}].$$

User may choose second one so that the stepper will first evaluate $5 + 6$ and put result into mark. Hence, we get $4 + 1 + 11$ finally.

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.

4 CONCLUSION

This paper develops an interactive stepper for Hazel, hence, language environment with holes. It uses the Paused judgement to detect the unnecessary situation so that the stepper will stop over the paused expression. Also, we develop a simple algorithm for decomposition that can find all

subexpressions which need to evaluate. We use the contextual dynamics to progress the expression for stepper. This result is useful for debugging or educational purpose.