

# Lifting Resugaring by Lazy Desugaring

ANONYMOUS AUTHOR(S)

## A APPENDIX

### A.1 Extension for black-box core evaluator

Discussion in this subsection is based on Section 3 of the main text. One may notice the traditional resugaring approach does not need the whole evaluation rules of core language, a black-box stepper is enough instead. Our approach can also work by given a black-box stepper with a tricky extension, but a little more information is needed. Here we introduce the extension firstly. We use  $\rightarrow_c$  to denote a reduction step of core language's expression in the black-box stepper, and  $\rightarrow_e$  to denote a step in the extension evaluator for the mixed language. We may use  $\rightarrow_m$  to denote the one-step reduction in our mixed language, defined in Section 3.2.

$$\frac{\forall i. e_i \in \text{CoreExp} \quad (\text{CoreHead } e_1 \dots e_n) \rightarrow_c e'}{(\text{CoreHead } e_1 \dots e_n) \rightarrow_e e'} \quad (\text{CORERED})$$

$$\frac{\forall i. tmp_i = (e_i \in \text{SurfExp} ? tmpe : e_i), \text{ where } tmpe \text{ is any reducible CoreExp term} \quad (\text{CoreHead } tmp_1 \dots tmp_i \dots tmp_n) \rightarrow_c (\text{CoreHead } tmp_1 \dots tmp'_i \dots tmp_n)}{(\text{CoreHead } e_1 \dots e_i \dots e_n) \rightarrow_e (\text{CoreHead } e_1 \dots e'_i \dots e_n) \quad \text{where } e_i \rightarrow_m e'_i \text{ if } e_i \in \text{SurfExp}, \text{ else } e_i \rightarrow_e e'_i} \quad (\text{COREEXT1})$$

$$\frac{\forall i. tmp_i = (e_i \in \text{SurfExp} ? tmpe : e_i), \text{ where } tmpe \text{ is any reducible CoreExp term} \quad (\text{CoreHead } tmp_1 \dots tmp_n) \rightarrow_c e' \text{ // not reduced in subexpressions}}{(\text{CoreHead } e_1 \dots e_n) \rightarrow_e e'[e_1/tmp_1 \dots e_n/tmp_n]} \quad (\text{COREEXT2})$$

Then we should replace the rules CoreRed1 and CoreRed2 by the following rule.

$$\frac{(\text{CoreHead } e_1 \dots e_n) \rightarrow_e e'}{(\text{CoreHead } e_1 \dots e_n) \rightarrow_m e'} \quad (\text{EXTRED})$$

Putting them in simple words. For expression  $(\text{CoreHead } e_1 \dots e_n)$  whose subexpressions contain SurfExp, replacing all SurfExp subexpressions not in core language with any reducible core language's term  $tmpe$ . Then getting a result after inputting the new expression  $e'$  to the original black-box stepper. If reduction appears at a subexpression at  $e_i$  or what the  $e_i$  replaced by, then the stepper with the extension should return  $(\text{CoreHead } e_1 \dots e'_i \dots e_n)$ , where  $e'_i$  is  $e_i$  after the mixed language's one-step reduction ( $\rightarrow_m$ ) or after core language's reduction with extension ( $\rightarrow_e$ ) (rule CoreExt1, an example in Figure 1). Otherwise, the stepper should return  $e'$ , with all the replaced subexpressions replacing back (rule CoreExt2, an example in Figure 2). The extension will not violate the properties of original core language's evaluator. It is obvious that the evaluator with the extension will reduce at the subexpression as it needs in core language, if the reduction appears in a subexpression. The stepper with extension behaves the same as mixing the evaluation

2018. 2475-1421/2018/1-ART1 \$15.00

<https://doi.org/>

rules of core language and desugaring rules of surface language. The extension is just to make it works when the evaluator of core language is a black-box stepper, by getting context rules using the tmpe. That's why the extension is tricky.

```

(if (and e1 e2) true false)
  ↓replace
(if tmpe1 true false)
  ↓blackbox
(if tmpe1' true false)
  ↓desugar
(if (if e1 e2 false) true false)

```

Fig. 1. CoreExt1's Example

```

(if (if true ture false) (and ...) (or ...))
  ↓replace
(if (if true ture false) tmpe2 tmpe3)
  ↓blackbox
(if true tmpe2 tmpe3)
  ↓replaceback
(if true (and ...) (or ...))

```

Fig. 2. CoreExt2's Example

But something goes wrong when substitution takes place during CoreExt2. For a expression like  $(\text{let } x \ 2 \ (\text{Sugar } x \ y))$  as an example, it should reduce to  $(\text{Sugar } 2 \ y)$  by the CoreRed2 rule, but got  $(\text{Sugar } x \ y)$  by the CoreExt2 rule. So when using the extension of black-box stepper's rule (ExtRed2), we need some other information about in which subexpression a substitution will occur (the substitution can be got by a similar idea as the tricky extension). Then for these subexpressions, we need to do the same substitution before replacing back. For example, we know the third subexpression of a expression headed with let is to be substituted. we should first try  $(\text{let } x \ 2 \ x)$ ,  $(\text{let } x \ 2 \ y)$  in one-step reduction to get the substitution  $[2/x]$ , then, getting  $(\text{Sugar } 2 \ y)$ .