A lightweight resugaring approach based on reduction semantics*

Subtitle†

ANONYMOUS AUTHOR(S)

With the rapid development of computer science, domain-specific language (DSL) is quite useful in our daily life, not only for programmers or computer scientists, but for people from all walks of life. Syntactic sugar is a good way to implement embedded DSLs, because it can make good use of existing general-purposed language's feature. However, the evaluation sequences became unrecognizable after the sugar expression desugared.

Resugaring is an method to solve the problem above. In this paper, we purposed a lightweight approach of resugaring based on reduction semantics—getting evaluation sequences without fully desugaring the whole syntactic sugar expression. We implement a tool based on our method using PLT Redex and test our approach on some applications. The results show that our lightweight approach can even deal with more syntactic sugar's feature.

Additional Key Words and Phrases: Domain-specific Language, Syntactic Sugar, Interpreter

1 INTRODUCTION

Domain-specific language[Fowler 2011] is becoming useful for people's daily tasks. For example, the IFTTT app and IOS's shortcuts designed DSLs describing some tasks to make our lives more convenient. So the users of DSL are no longer limited to programmers, but people from all walks of life.(to be completed)

Syntactic sugar[Landin 1964], as a simple ways design DSL, has a obvious problem. DSL based on syntactic sugars contains many components of its host language. Then its interpretation will be outside the DSL itself. The evaluation sequences of syntactic sugar expression will contain many terms of the host language, which may confuse the users of DSL.

There is an existing work—resugaring[Pombrio and Krishnamurthi 2014][Pombrio and Krishnamurthi 2015], which aimed to solve the problem upon. It lifts the evaluation sequences of desugared expression to sugar's syntax. The evaluation sequences shown by resugaring will not contain components of host language. But we found the resugaring method using match and substitution is kind of redundant. The biggest deficiency of existing resugaring method is that the syntactic sugars in an expression have to fully desugar before evaluation. This limits the processing ability of the method. Moreover, it limits the complexity of getting the resugaring sequences. If we need to resugar a very huge expression, the match and substitution processes will cost so much. Also, processing of hygienic macros is complex due to the extra data structure.

In this paper, We propose a lightweight approach to get resugaring sequences based on syntactic sugars. The key idea of our approach is—syntactic sugar expression only desugars at the point that it have to desugar. We guess that we don't have to desugar the whole expression at the initial time of evaluation under the premise of keeping the properties of expression.

Initially, our work focused on improving current resugaring method. After finishing that, we found our lightweight resugaring approach could process some syntactic sugars' feature that current approach cannot do. Finally, we implement our algorithm using PLT Redex[Felleisen et al.

*Title note
†Subtitle note

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/

2009] and test our approach on some applications. The result shows that our approach does handle more features of syntactic sugar.

In the rest of this paper, we present the technical details of our approach together with the proof of correctness. In details, the rest of our paper is organized as follow:

- An overview of our approach with some background knowledge.[sec 2]
- The algorithm defination and proof of correctness.[sec 3]
- The implementation of our lightweight resugaring algorithm using PLT Redex.[sec 4]
- sth else?[sec 5]
- Evaluation of our lightweight resugaring approach.[sec 6]

2 OVERVIEW

Use a simple but sharp example to give an overview of your approach.

2.1 Defination of resugaring

This subsection is partially similiar to original defination in[Pombrio and Krishnamurthi 2014].

Defination 2.1 (Resugaring). *Given core language (named **CoreLang**) and its evalation rules, together with surface language based on syntactic sugars of CoreLang (named **Surflang**). For any expression of Surflang, getting the evaluation sequences of the expression in terms of SurfLang.*

For correctness of the resugaring, the evaluation sequences should maintain the following three properties:

(1) **Emulation** Each term in the generated surface evaluation sequence desugars into the core term which it is meant to represent.
(2) **Abstraction** The resugaring sequences should only contains terms in SurfLang, and each term of SurfLang should originate from initial expression.
(3) **Coverage** No sequence is skipped during the process.

Given an example below.
For syntactic sugar **and** and **or**, the sugar rules are:

$$\text{and(e1, e2)} \longrightarrow \text{if(e1, e2, \#f)}$$
$$\text{or(e1, e2)} \longrightarrow \text{if(e1, \#t, e2)}$$

which forms a simple SurfLang.
The evaluation rules of **if** is:

$$\text{if(\#t, e1, e2)} \longrightarrow \text{e1}$$
$$\text{if(\#f, e1, e2)} \longrightarrow \text{e2}$$

Then for SurfLang's expression and(or($\#f$, $\#t$), and($\#t$, $\#f$)) should get resugaring sequences as fig1.todo:why

2.2 Idea origin

Church–Rosser theorem[Church and Rosser 1936] gives theoretical support for full-β reduction, which is a nondeterministic evaluation strategy of lambda caculus.

Reduction semantics

Our original idea is similiar to full-β reduction. When not restricting the context rules of reduction semantics, the reduction paths of a expression will become a full graph like full-β reduction.

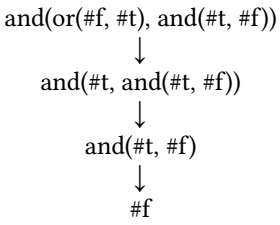

Fig. 1. resugaring example

3 LIGHTWEIGHT ALGORITHM

3.1 Language setting

3.1.1 *Grammatical restrictions.*

Firstly, the whole language should restrict to tree-structured disjoint expression.

Defination 3.1 (disjoint). *For every sub-expression in a expression, its reduction rule is decided by itself.*

This restriction limits the scope of language. Every sub-expression must have no side effect. We will discuss more on side effect in ...

Defination 3.2 (tree-structured). *The grammar of the whole language is defined as follow.*

$$\begin{array}{rcl} \mathit{Exp} & ::= & (\mathit{Headid}\ \mathit{Exp}*) \\ & | & \mathit{Value} \\ & | & \mathit{Variable} \end{array}$$

The grammatical restrictions give our language a similiar property as church-rosser theorem for lambda calculus.

todo:church-rosser?

3.1.2 *Context restrictions.*

For expressions in CoreLang, the context rule should restrict it to have only one reduction path. The context rules can limit the order of evaluation. This restriction is normal, because a program in general-purposed language should have only one execution path.

For expressions in SurfLang, context rules should allow every sub-expressions reduced. It's the same as full-β reduction.

3.1.3 *Restriction of syntactic sugar.*

The form of syntactic sugar is as follow. $\boxed{(\text{Surfid}\ e_1\ e_2\ \ldots) \rightarrow (\text{Headid}\ \ldots)}$

An counter example of this restriction is (Surfid $\ldots$ $(e1\ e2)\ldots$)) in LHS. It's for simpler algorithm form, and the expression ability of syntactic sugar will not be changed.

Defination 3.3 (Unambiguous). *For every syntactic sugar expression, it can only desugar to one expression in CoreLang.*

3.1.4 *Grammar Description.*

In our language setting, we regard SurfLang and CoreLang as a whole language. The whole language is under restrictions above, and its grammar is defined as follow.

```
Exp ::= DisplayableExp
      | UndisplayableExp
DisplayableExp ::= Surfexp
                 | Commonexp
UndisplayableExp ::= Coreexp
                   | OtherSurfexp
                   | OtherCommonexp
Coreexp ::= (CoreHead Exp*)
Surfexp ::= (SurfHead DisplayableExp*)
Commonexp ::= (CommonHead DisplayableExp*)
            | Value
            | Variable
OtherSurfexp ::= (SurfHead Exp * UndisplayableExp Exp*)
OtherCommonexp ::= (CommonHead Exp * UndisplayableExp Exp*)
```

The difference between CoreLang and SurfLang is identified by *Headid*. But there are some terms in CoreLang should be displayed during evaluation. Or we need some terms to help us getting better resugaring sequences. So we defined **Commonexp**, which origin from CoreLang, but can be displayed in resugaring sequences. The **Coreexp** terms are terms with undisplayable CoreLang's Headid. The **Surfexp** terms are terms with SurfLang's Headid and all sub-expressions are displayable. The **Commonexp** terms are terms with displayable CoreLang's Headid, together with displayable sub-expressions. There exists some other expression during our resugaring process. They have Headid which can be displayed, but one or more subexpressions can't. They are UndisplayableExp.

3.2 Algorithm defination

Our lightweight resugaring algorithm is based on a core algorithm f. For every expression during resugaring process, it may have one or more reduction rules. The core algorithm f chooses the one that satisfies three properties of resugaring, then applies it on the given expression. The core algorithm f is defined as 1.

We briefly describe the core algorithm f in words.

For Exp in language defined as last section, try all reduction rules in the language, get a list of possible expressions ListofExp'=$\{Exp'_1, Exp'_2, \ldots\}$.

Line 2-9 deal with the case when Exp has a CoreLang's Headid. When Exp is value or variable (line 3-4), ListofExp' won't have any element (not reducible). When Exp is of Coreexp or Commonexp (line 5-6, due to the context restriction of CoreLang, only one reduction rule can be applied. When Exp is OtherCommonexp (line 7-8), due to the context restriction of CoreLang, only one sub-expression can be reduced, then just apply core algorithm recursively on the sub-expression.

Line 10-21 deal with the case then Exp has a SurfLang's Headid. When Exp only has one reduction rule (line 11-12), the syntactic sugar has to desugar. If not, we should expand outermost sugar and find the sub-expression which should be reduced (line 14-16), or the sugar has to desugar (line 17-18).

Algorithm 1 Core-algorithm f

Input:

Any expression Exp=($Headid\ Subexp_1\ \ldots\ Subexp_{\ldots}$) which satisfies Language setting

Output:

Exp' reduced from Exp, s.t. the reduction satisfies three properties of resugaring

```
1:  Let ListofExp' = {Exp'_1 , Exp'_2 ...}
2:  if Exp is Coreexp or Commonexp or OtherCommonexp then
3:    if Lengthof(ListofExp')==0 then
4:      return null; //                                                    Rule1.1
5:    else if Lengthof(ListofExp')==1 then
6:      return first(ListofExp'); //                                       Rule1.2
7:    else
8:      return Exp'_i = (Headid Subexp_1 ... Subexp'_i ...); //where i is the index of subexp which
        have to be reduced.                                               Rule1.3
9:    end if
10: else
11:   if Exp have to be desugared then
12:     return desugarsurf(Exp); //                                       Rule2.1
13:   else
14:     Let DesugarExp = desugarsurf(Exp)
15:     if Subexp_i is reduced to Subexp'_i during f(DesugarExp) then
16:       return Exp'_i = (Headid Subexp_1 ... Subexp'_i ...); //          Rule2.2.1
17:     else
18:       return DesugarExp; //                                           Rule2.2.2
19:     end if
20:   end if
21: end if
```

If Exp is Coreexp or Commonexp or OtherCommonexp, then

- ListofExp' may be empty, the Exp is not reducible. Return Empty then. *Rule*1.1
- ListofExp' may contain only one element. Just return it because no more reduction available. *Rule*1.2
- ListofExp' may contain more than one elements. Due to context restriction of CoreLang3.1.2, only one sub-expression $Subexp_i$ can be reduced. Now we apply core language f on $Subexp_i$ recursively to get $Subexp'_i$, and return the expression in which $Subexp_i$ is reduced to $Subexp'_i$. *Rule*1.3

If Exp is Surfexp or OtherSurfexp, then

- ListofExp' may contain only one element. Just return it because no sub-expression can be reduced, The syntactic sugar of Headid have to desugar. *Rule*2.1
- ListofExp' may contain more than one elements. Due to unambiguous restriction of syntactic sugar, there must exist sub-expression which can be reduced. Firstly, we desugar **Exp**'s outermost syntactic sugar to **DesugarExp**. Then apply core algorithm f on **DesugarExp**. (named one-step try)
  - If f(DesugarExp) reduces sub-expression of DesugarExp, since the sub-expression is composed by sub-expressions of **Exp**, it is necessary to detect which sub-expression $Subexp_i$ is

first reduced to approximately $Subexp'_i$. Return the expression in which $Subexp_i$ is reduced to $Subexp'_i$. *Rule*2.2.1

– If f(DesugarExp) doesn't reduce sub-expression of DesugarExp, then the outermost sugar won't resugar. Return **DesugarExp** then. *Rule*2.2.2

Then, our lightweight-resugaring algorithm is defined as 2.

Algorithm 2 Lightweight-resugaring

Input:
Surfexp *Exp*
Output:
Exp's evaluation sequences within DSL

```
1: while tmpExp = f(Exp) do
2:   if tmpExp is empty then
3:     return
4:   else if tmpExp is Surfexp or Commonexp then
5:     print tmpExp;
6:     Lightweight-resugaring(tmpExp);
7:   else
8:     Lightweight-resugaring(tmpExp);
9:   end if
10: end while
```

3.3 Proof of correctness

First of all, because the difference between our lightweight resugaring algorithm and the existing one is that we only desugar the syntactic sugar when needed, and in the existing approach, all syntactic sugar desugars firstly and then executes on CoreLang.

Second, to prove convenience, define some terms.

$Exp = (Headid\ Subexp_1\ Subexp_{...} \ldots)$ is any reducible expression in our language.

If we use the reduction rule that desugar Exp's outermost syntactic sugar, then the reduction process is called **Outer Reduction**.

If the reduction rule we use reduce $Subexp_i$, where $Subexp_i$ is $(Headid_i\ Subexp_{i1}\ Subexp_{i...} \ldots)$

- If the reduction process is Outer Reduction of $Subexp_i = (Headid_i\ Subexp_{i1}\ Subexp_{i...} \ldots)$, then it is called **Surface Reduction**.
- If the reduction process reduces $Subexp_{ij}$, then it is called **Inner Reduction**.

Example:

$(\text{if } \#t\ Exp_1\ Exp_2) \rightarrow Exp1$ Outer Reduction

$(\text{if } (\text{And } \#t\ \#f)\ Exp_1\ Exp_2) \rightarrow (\text{if } (\text{if } \#t\ \#f\ \#f)\ Exp_1\ Exp_2)$ Surface Reduction

$(\text{if } (\text{And } (\text{And } \#t\ \#t)\ \#t)\ \#f)\ Exp_1\ Exp_2) \rightarrow (\text{if } (\text{And } \#t\ \#t)\ Exp_1\ Exp_2)$ Inner Reduction

Defination 3.4 (Upper and lower expression). *For Exp=(Headid* $Subexp_1$ $Subexp_{...} \ldots$*), Expis called* ***upper expression****,* $Subexp_i$ *is called* ***lower expression****.*

Proof of Emulation.

□

Proof of Abstraction. 2 □

Proof of Coverage. 3 □

4 CONTRIBUTION2 ...

Explain your second technical contribution.

5 CONTRIBUTION3 ...

Explain your third technical contribution.

6 EVALUATION

Explain how your system is implemented and how the experiment is performed to evaluate your approach.

7 RELATED WORK

Explain the work that are related to your problem, and to your three contributions.

8 CONCLUSION

Summarize the paper, explaining what you have shown, what results you have achieved, and what future work is.

REFERENCES

Alonzo Church and J. B. Rosser. 1936. Some Properties of Conversion. *Trans. Amer. Math. Soc.* 39, 3 (1936), 472–482. http://www.jstor.org/stable/1989762

Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. 2009. *Semantics Engineering with PLT Redex* (1st ed.). The MIT Press.

Martin Fowler. 2011. *Domain-Specific Languages.* Addison-Wesley. http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0321712943,00.html

P. J. Landin. 1964. The Mechanical Evaluation of Expressions. *Comput. J.* 6, 4 (01 1964), 308–320. https://doi.org/10.1093/comjnl/6.4.308 arXiv:https://academic.oup.com/comjnl/article-pdf/6/4/308/1067901/6-4-308.pdf

Justin Pombrio and Shriram Krishnamurthi. 2014. Resugaring: Lifting Evaluation Sequences through Syntactic Sugar. *SIGPLAN Not.* 49, 6 (June 2014), 361–371. https://doi.org/10.1145/2666356.2594319

Justin Pombrio and Shriram Krishnamurthi. 2015. Hygienic Resugaring of Compositional Desugaring. *SIGPLAN Not.* 50, 9 (Aug. 2015), 75–87. https://doi.org/10.1145/2858949.2784755

A APPENDIX

Text of appendix ...