A lightweight resugaring method 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 approach to solve the problem above. In this paper, we purposed a lightweight mothod 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

What is the remained research problem and how challenge it is?

What are the three main technical contributions of this paper?

The rest of the paper is organized as follows. ...

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.

*Title note

†Subtitle note

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

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. 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.2
- The algorithm defination and proof of correctness.3
- The implementation of our lightweight resugaring algorithm using PLT Redex.4
- sth else?5
- Evalation of our lightweight resugaring approach.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 (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:

- and(e1, e2) → if(e1, e2, #f)
- or(e1, e2) → if(e1, #t, e2)

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

- if(#t, e1, e2) → e1
- if(#f, e1, e2) → e2

3 ALGORITHM

The goals of our dynamic approach is similar with Resugaring[Pombrio and Krishnamurthi 2014][Pombrio and Krishnamurthi 2015], that is, get evaluation sequences of surface language's expression using surface expression's syntax. However, their approach make it by converting evaluation sequences of desugared expressions into surface language's expression, using match and substitution on syntactic sugars' rule. We do this by**not expanding syntactic sugar until necessary**. Our approach shows some better properties for implementing DSL.

3.1 Language setting

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

$$
\begin{array}{rcl}
\text{Exp} & ::= & \text{Coreexp} \\
& | & \text{Surfexp} \\
& | & \text{Commonexp} \\
& | & \text{OtherSurfexp} \\
& | & \text{OtherCommonexp} \\
\\
\text{Coreexp} & ::= & (\text{CoreHead Exp}*) \\
\\
\text{Surfexp} & ::= & (\text{SurfHead (Surfexp | Commonexp)}*) \\
\\
\text{Commonexp} & ::= & (\text{CommonHead (Surfexp | Commonexp)}*) \\
& | & \text{Value} \\
& | & \text{Variable} \\
\\
\text{OtherSurfexp} & ::= & (\text{SurfHead Exp} * \text{ Coreexp Exp}*) \\
\\
\text{OtherCommonexp} & ::= & (\text{CommonHead Exp} * \text{ Coreexp Exp}*)
\end{array}
$$

3.2 Algorithm defination

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

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

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

Let $ListofExp' = \{Exp'_1, Exp'_2\ \ldots\}$
if Exp is Coreexp or Commonexp or OtherCommonexp **then**
  if Lengthof($ListofExp'$)==0 **then**
    return null; // Rule1.1
  else if Lengthof($ListofExp'$)==1 **then**
    return first($ListofExp'$); // Rule1.2
  else
    return Exp'_i = ($Headid\ Subexp_1\ \ldots\ Subexp'_i\ \ldots$); //where i is the index of subexp which have to be reduced. Rule1.3
  end if
else
  if Exp have to be desugared **then**
    return desugarsurf(Exp); // Rule2.1
  else
    Let $DesugarExp'$ = desugarsurf(Exp)
    if $Subexp_i$ is reduced to $Subexp'_i$ during $f(DesugarExp')$ **then**
      return Exp'_i = ($Headid\ Subexp_1\ \ldots\ Subexp'_i\ \ldots$); // Rule2.2.1
    else
      return desugarsurf(Exp); // Rule2.2.2
    end if
  end if
end if

Algorithm 2 Lightweight-resugaring

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

while $tmpExp$ = f(Exp) **do**
  if $tmpExp$ is empty **then**
    return
  else if $tmpExp$ is Surfexp or Commonexp **then**
    print $tmpExp$;
    Lightweight-resugaring($tmpExp$);
  else
    Lightweight-resugaring($tmpExp$);
  end if
end while

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 …