Dynamic and Static Interpreter for Domain-specific Language Based on Syntactic Sugars*

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, it's regerttable that DSL based on syntactic sugars cannot be called "programming language"— it does not even have its own interpreter. Its interpreter, though existed, contains many things about its host language.

In this paper, we focus on get a better interpreter for DSL based on syntactic sugars, from two different approachs—dynamic, which get evaluation sequences of DSL's expression within DSL itself; static, which get evaluation rules of DSL's expression within DSL itself. We also build a tool to test our approach with some interesting applications.

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. It will be better if we can get interpreter of DSL without host language's components, when given host language's interpreter and DSL's rewriting rules. It's important for DSL becoming a real programming language—its interpreter should not depend on something outside itself. Then the real language will be more concise during execution, or some program analysis, optimization tasks.

There is an existing work—resugaring[Pombrio and Krishnamurthi 2014][Pombrio and Krishnamurthi 2015], which partially solved 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. Resugaring also cannot perfectly solve some syntactic sugar's feature, such as recursive sugar, higher-order sugar. (todo: static approachs' advantages)

(challenge)

We propose two diffenent approachs to get better interpreters for DSLs based on syntactic sugars. Initially, our work focused on improving current resugaring method(dynamic approach). After

*Title note

†Subtitle note

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

finishing that, we found the dynamic approach could be abstracted to evalation rules of DSL(static approach).

The key idea of the dynamic 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. (todo: static approach insight)

2

2 OVERVIEW

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

3 DYNAMIC APPROACH

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

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

$$
\boxed{
\begin{array}{rcl}
\text{Exp} & ::= & \text{Coreexp} \\
& | & \text{Surfexp} \\
& | & \text{Commonexp} \\
& | & \text{OtherSurfexp} \\
& | & \text{OtherCommonexp} \\[1ex]
\text{Coreexp} & ::= & (\text{CoreHead Exp}*) \\[2ex]
\text{Surfexp} & ::= & (\text{SurfHead } (\text{Surfexp} \mid \text{Commonexp})*) \\[2ex]
\text{Commonexp} & ::= & (\text{CommonHead } (\text{Surfexp} \mid \text{Commonexp})*) \\
& | & \text{Value} \\
& | & \text{Variable} \\[1ex]
\text{OtherSurfexp} & ::= & (\text{SurfHead Exp} * \ \text{Coreexp Exp}*) \\[2ex]
\text{OtherCommonexp} & ::= & (\text{CommonHead Exp} * \ \text{Coreexp Exp}*)
\end{array}
}
$$

3.2 Algorithm

4 CONTRIBUTION2 ...

Explain your second technical contribution.

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

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

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 ...