北京大學

本科生毕业论文

题目： 一种利用 **Redex** 实现重组糖的轻量级方法

A lightweight resugaring method using PLT Redex

姓　　名：杨子毅

学　　号：1600011063

院　　系：信息科学技术学院

本科专业：软件工程

指导老师：胡振江

二〇二〇　年　五　月

北京大学本科毕业论文导师评阅表

<table>
<tr><td>学生姓名</td><td></td><td>学生学号</td><td></td><td>论文成绩</td><td></td></tr>
<tr><td>学院 (系)</td><td colspan="3"></td><td>学生所在专业</td><td></td></tr>
<tr><td>导师姓名</td><td></td><td>导师单位/
所在研究所</td><td></td><td>导师职称</td><td></td></tr>
<tr><td colspan="2">论文题目
（中、英文）</td><td colspan="4"></td></tr>
<tr><td colspan="6">导师评语
（包含对论文的性质、难度、分量、综合训练等是否符合培养目标的目的等评价）

导师签名：

年　　　月　　　日</td></tr>
</table>

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任。

摘要

随着计算机科学的普及和编程语言的发展，编程语言、特别是领域特定语言的应用越来越日常化。语法糖作为实现领域特定语言的一项重要技术在近年来发展火热，相关的研究[1]以及为 DSL 而诞生的编程语言 (例如 Racket[1]) 都在进展显著。

我们对一个重组糖[1]（语法糖的一项工作）系列的文章进行学习，并基于 PLT Redex 设计了一个轻量级重组糖算法，简单实现了一套工具并在一些例子上进行测试。结果显示我们的轻量级重组糖算法相较于现有重组糖算法可以多处理一些语法糖特性，也更容易处理卫生宏等特性。除此之外，由于该方法在某种意义上是对领域特定语言的一种解释执行，因此对自动生成领域特定语言的解释器研究有一定参考价值。

关键词: 领域特定语言、语法糖、重写系统、函数式语言、解释器

Abstract

With the popularization of computer science and the development of programming languages, the application of programming languages, especially domain-specific languages, is becoming more and more routine. Syntactic sugar, as an important technique for implementing a domain-specific language, has developed fiercely in recent years, and related research and programming languages born for DSL(such as Racket) are making significant progress.

We studied a series of articles on Resugaring (a work of syntactic sugar), and designed a lightweight resugaring algorithm based on PLT Redex, simply implemented a set of tools and tested on some examples. The results show that our lightweight resugaring algorithm can handle more syntactic sugar features than existing resugaring algorithms, also handle features like hygienic macro more simply. In addition, because the method is a kind of interpretation of domain-specific language in a sense, it has a certain reference value for the research of automatic generating the interpreter of domain-specific language.

Key Words: Domain-specific language, syntactic sugar, rewriting system, functional language, interpreter

全文目录

第一章 绪论

1. 研究背景

领域特定语言的研究及应用日益广泛，其中不乏正则表达式、SQL、XML等应用场合及其多的领域特定语言。而随着计算机科学技术的发展，更多新颖的的场景（引用）加入到了使用 DSL 的队伍。而在一般场合中，语言设计者是计算机科学家，语言使用者却是领域内的专家。因此，当领域特定语言的使用出现一些问题时，领域内的专家（使用者）需要得到领域特定的信息来进行处理。

语法糖是近些年一种流行的领域特定语言实现方法，其方法源于 Lisp 的宏系统（Macro），经过 Scheme 语言的发展，再到 Racket 语言的扩展。（可以引用 From …那篇的部分）语法糖构造 DSL 的最主要优点就是简单高效，语言设计者只需要写一个简单的映射，就可以构造 DSL。

然而，语法糖的一项缺陷，导致其应用领域一直局限于计算机科学内部。我们先来看下面这个例子。

（举一个开药方/饭店流程的例子）

我们可以看到，在执行过程中，语法糖被展开成 if，在通用语言中继续执行，得到最终结果。但实际应用到特定领域时，我们很明显不希望得到这样复杂的执行过程，特别是对于对计算机内部语言不熟悉的领域专家来说，这种执行过程是没有意义的。

2. 解决的问题及研究现状

我们将上述问题总结为语法糖解糖的单向性，对上面的例子的 xx 行 todo，我们可以看出，其存在等价的领域特定语言表示 and(1,and(1,0))。如果语法糖的解糖有一个逆过程（重组糖），我们就可以找到其对应的这样一个序列。我们将在第二章对问题进行形式化定义。我们在后文，将领域特定语言视为外部语言，通用语言视为内部语言。

3. 相关工作及存在的问题

我们的方法主要借鉴和对比 Resugaring 系列一些工作，其中前两篇和本文的工作紧密相关，我们将简单讲述一下这两篇工作的方法及缺陷

第一篇介绍的方法，基本思想是将内部语言的求值序列每一步加上标签，进行搜索，试图得到其对应的在外部语言的表示。

其算法希望具有如下三个性质：

仿真性/抽象性/覆盖性（没有被证明）

第二篇工作在第一篇的基础上，新增了两个优点：1. 解决卫生宏 2. 拓展语法糖规则（超过 syntax-rule）3. 覆盖性得到形式化证明

然而该工作仍然存在一些问题：

1. 语法糖规则依然不够丰富

2. 算法定义繁琐，通用性差。

4. 基本思路及全文结构

本文基于语义工程工具 PLT Redex，设计了一个全新的语法糖——重组糖框架。其想法源于完全 β 规约的完备性，如下图。

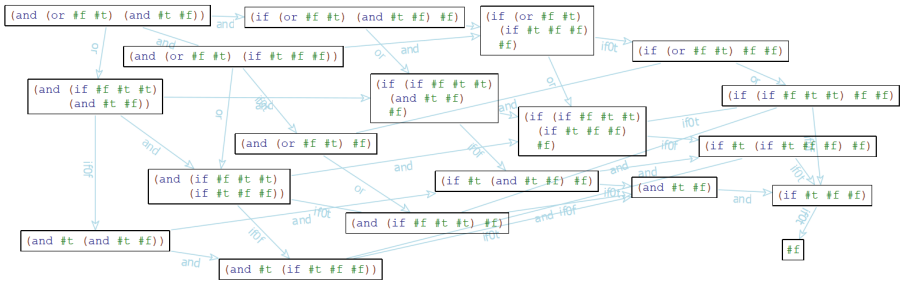

图 1 基本思路

该例子中，对一个结构化（类似 S 表达式，定义在第二章）的语言，定义其基础语义和规约规则。其中 and 和 or 是语法糖，语法糖展开内部语言的 if 表达式。在没有限制其上下文规则（求值顺序）的前提下，生成了其规约的流程图。我们可以看出，在该图中，既包含了将语法糖展开的规约，也包含了我们期待的 and() 等等这些中间求值路径。因此我们希望使用基于规约语义的 PLT Redex，实现一个轻量级的重组糖，在这个完全图中提取出我们需要的重组糖序列。

在实现过程中，我们发现生成中国完全图并不是必须的，而我们可以在从最

初的表达式开始**每次进行单步规约，在一条或多条规约规则中选择我们需要的那条规约规则** (是本工作的核心算法)，且该规则的多次执行保证重组糖的三个基本重要性质。则在这个核心算法迭代执行过程中，会留下一个对应的求值序列，在其中提取出符合输出规则的中间序列，则此序列即为重组糖的输出。

我们将在第二章讲述本文工作的一些背景知识及思考路线，第三章讲述工作的算法定义及正确性证明，第四章讲述一些实现算法的具体细节及对一些语法糖例子进行讨论评估，第五章总结我们的工作，并对一些未来可能的方向进行简单探讨与展望。

5. 本文主要贡献

1. 我们针对现有重组糖的方法法进行改进，得到新的轻量级重组糖方法。

- 我们的方法不需要将所有语法糖都展开就可以得到重组糖序列，而现有方法需要展开后在内部语言执行，并基于 match 和 substitute 对可重组的语法糖进行搜索。
- 我们的方法处理卫生宏很简单而自然，而现有工作处理卫生宏需要引入新的数据结构以及很多其他处理。

2. 我们的方法相对于现有重组糖方法，支持更多语法糖特性。

- 对递归糖，我们的方法可以很简单的处理。而现有方法只能用 letrec 处理一下递归绑定。
- 对高阶糖，我们的方法也可以很容易处理。而现有方法不能处理。

3. 我们的轻量级重组糖算法在对领域特定语言的解释器进行自动生成的研究提供了一个思路。我们将在未来工作中进一步讨论。

第二章 背景知识

1. 重组糖形式化定义

对于给定求值规则的内部语言 CoreLang，和在 CoreLang 基础上用语法糖构造的表面语言 SurfLang；对于任意 SurfLang 的表达式，得到其在 SurfLang 上的求值序列，且该求值序列满足三个性质：

1. 仿真性：求值序列需要和在 CoreLang 上的求值顺序相同，即存在 CoreLang 上的求值序列中的部分中间过程与该序列中的元素对应。该性质是重组糖有意义的前提。

2. 抽象性：求值序列中只存在 SurfLang 中存在的术语，没有引入 CoreLang 中的术语。该性质是重组糖研究的目的。

3. 覆盖性：在求值序列中没有跳过一些中间过程。该性质不是正确性的必要条件，却是在应用中极其重要的；加上前两条性质满足的正确性，构成了重组糖的全部重要性质。

例子：

$$and(or(\#f, \#t), and(\#t, \#f)) \tag{2.1}$$

语法糖规则 (Surflang)：

$$\begin{gathered}\text{and(e1,e2)==if(e1,e2,\#f)} \\ \text{or(e1,e2)==if(e1,\#t,e2)}\end{gathered} \tag{2.2}$$

其中 if 的规则是在 coreLang 上规定的，其具体规约规则如下：

$$\begin{gathered}\text{if(\#t,e1,e2)==e1} \\ \text{if(\#f,e1,e2)==e2}\end{gathered} \tag{2.3}$$

则我们期望得到的重组糖序列是如下的序列:

```
and(or(#f,#t),and(#t,#f))–>
                              and(#t,and(#t,#f))
                              and(#t,#f)
                              #f
```
(2.4)

全文将围绕这个例子展开初步的讲解。

2. 完全 β 规约、归约语义和 PLT Redex

与 β 规约的概念不同，完全 β 规约是一种基于 β 规约的求值顺序规则。对于一个嵌套的 lambda 表达式，每个表达式都可能进行 β 规约，而常规的 call-by-name 和 call-by-value 都是对规约顺序进行了约定，而完全 β 规约就是一种不定序的求值规则，每个可 β 规约的位置都有可能进行规约，因此得到的规约路径不是一条，而是一个图，且这个图的起点和终点只有一个。如图所示的例子就是一个完全 β 规约的求值图

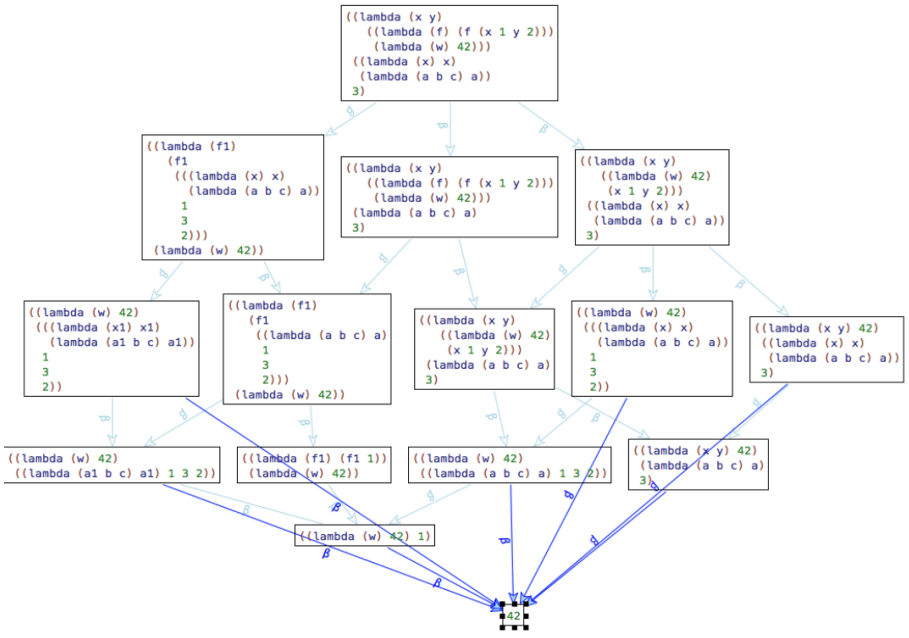

图 1 完全 β 规约

可以看出，在完全 beta 规则中，对任何位置的可 beta 规约的 lambda 表达式，都可以进行规约。因此，与 call-by-name 和 call-by-value 不同的是，这种求值规则是不定序的。

规约语义：我们需要在求值规则中约定每一个表达式的规约规则，。。。PLT Redex [2] 是基于此语义的语义工程工具。todo

本文工作的最初思想就是基于完全 beta 规约。我们在对规约语义不限制上下文环境的情况下，其规约路径也将成为类似完全 β 规约的图。还是基于上文的例子 and(or(#f,#t),and(#t,#f))，我们可以得到如下的完全规约图2。

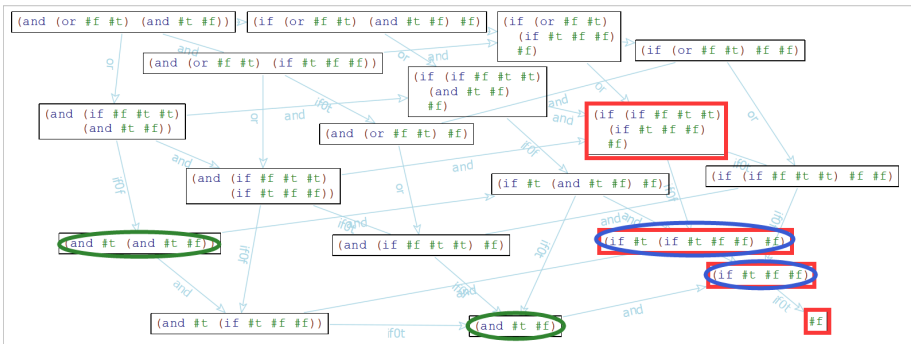

图 2 完全规约

在这个图中，我们可以看出，用红色标出的子序列是将语法糖直接展开后进行规约的求值序列，而这其中有许多中间表达式是可以重组成语法糖的，用蓝色标出。我们可以发现，在这个中间表达式中可重组的部分就是 (2.3) 中的重组糖序列。而又可以在图中找到绿色标出的序列—可以惊喜而又自然的发现这一条规约规约路径包含了我们想要的重组糖序列。自然是因为我们对语法糖的规约做了类似完全 β 规约的处理，导致每个子表达式都有可能首先被规约处理，因此重组糖序列一定在我们的完全图中。

第三章　算法定义及正确性证明

1. 对语言的规定

首先，我们需要将整个语言限定在基于树形表达式的结构化语言。

树形表达式：此处我们使用类似 Lisp 的 S 表达式的递归树，基础定义如下。

```
Exp::=
        (Headid Exp*)
        |Value
        |Variable
```

结构化：对于每个表达式中的子表达式，其规约规则只和子表达式本身有关。此限制约束了 Corelang 的作用域，限制语言子表达式不能有对外的副作用。我们将在第五章详细讨论副作用的一些具体解决办法

此外：我们对 CoreLang 和 SurfLang 进行一些简单的限制。

对每个子表达式都是 CoreLang 表达式的表达式 Exp，最多只能有一条规约路径（通过求值顺序约束）。这一点约束并不过分，为了保证每个程序只能有一条执行路径。

对 SurfLang，任意一个语法糖只能有一个 CoreLang 的表达式与之对应。这也是很自然的要求，因为同一个语法糖不应该有二义性。对于求值顺序，SurfLang 上的子表达式约定类似完全规约的规则，任何子表达式都可以首先进行规约。

最后: 对于语法糖的形式限制为如下。

$(Surfid\ e_1\ e_2 \ldots) --> (Id\ \ldots)$ 其中的主要约束是在语法糖这一侧不能出现形如 $(Surfid\ \ldots\ (e1\ e2)\ldots))$ 这种形式。这对语法糖的表达能力并不会造成影响，当然确实一定程度上限制了语法糖的形式。

在 PLT Redex 中，我们将 CoreLang 和 SurfLang 视为同一个语言。则当我们定义了一个语言内部各种规约规则后，对于任意 Exp 都有其对应的一条或多条规约规则。根据对 CoreLang 的约定，有多条规约规则的表达式必然存在 SurfLang 的表达式。

为了区分 CoreLang 的语言和 SurfLang 的语言，我们将表达式的文法定义为如下

```
Exp::=
        Coreexp
        |Surfexp
        |Commonexp
        |OtherSurfexp
        |OtherCommonexp
Coreexp ::= (CoreHead Exp*)
Surfexp ::= (SurfHead (Surfexp|Commonexp)*)
Commonexp::=
                (CommonHead (Surfexp|Commonexp)*)
                |value
                |variable
OtherSurfexp ::= (SurfHead Exp* Coreexp Exp*)
OtherCommonexp ::= (CommonHead Exp* Coreexp Exp*)
```

在这里，我们将 CoreLang 的表达式一部分提取处理作为公共表达式，是因为在重组糖序列中必定有一些表达式是属于 CoreLang 的，但需要在序列中输出（比如说数字，布尔表达式，以及一些可能的基础运算）。在这种情况下，对于 Commonexp 来说，满足 CoreLang 的约束，但是也可以作为重组糖的中间序列输出

可以看出，在我们的重组糖方法中，可以输出的表达式是 Surfexp 和 Commonexp，即不存在任何子表达式中存在 Coreexp。

小结：todo

2. 算法描述

本节讨论建立在符合约定的语言基础上。

核心算法 f[1] 定义如下：（输入为一个任意 Exp，输出为应用应该执行的规约规则后的表达式）

对 Exp 尝试所有规约规则，得到多个可能的表达式 ListofExp'=$\{Exp'_1, Exp'_2, \ldots\}$

1. 如果 Exp 是 Coreexp 或 Commonexp 或 OtherCommonexp，则其规约规则

- 或是将表达式规约到另一个表达式，此时只有一条规则，应用后输出 Exp';

Rule1.1

- 或是其规约不满足导致内部子表达式需要规约，此时因为 CoreLang 的定序性，只会有一个子表达式被规约（且此表达式为 Surfexp），此时对该子表达式 Subexp 递归调用核心算法 f 得到 Subexp'，则在 ListofExp' 中找到将此 Exp 中子表达式 Subexp 规约为 Subexp' 的表达式就是我们需要的表达式；

Rule1.2

- 或是已经无法被规约（ListofExp' 为空），此时返回的表达式为空。

Rule1.3

接下页

[1]核心思想：对于每个表达式 Exp，我们将对它所有规约规则中选择一条符合 resugaring 的仿真性规则的规约，且尽可能不破坏任何语法糖。

2. 如果 Exp 是 Surfexp 或 OtherSurfexp

- 如果内部子表达式无可规约的，则必然会展开该语法糖，此时输出表达式为 Exp 解糖后的表达式;

  Rule2.1

- 如果存在可规约的子表达式对于每个子表达式，如果可规约，则根据我们的设定，存在一条关于此子表达式的规约规则。因此每个子表达式都可能被规约的前提下，我们需要对 Surfexp 或 OtherSurfexp 的语法糖进行展开为 DesugarExp’（此展开只有一种规约规则对应），之后对 Exp’ 调用核心算法 f

  – 如果 f(DesugarExp’) 是对 DesugarExp’ 的子表达式进行规约，由于此子表达式一定由 Exp 的子表达式组成，需要检测是哪一个子表达式 $Subexp_i$ 先被规约为 $Subexp'_i$，则在 ListofExp’ 中将此子表达式规约的表达式就是所需要的。

    Rule2.2.1

  – 如果 f(DesugarExp’) 不是对子表达式进行规约，则说明这个糖不会被重组（此糖被展开后必然会被继续破坏），因此输出不在 ListofExp’ 中，而是输出 DesugarExp’

    Rule2.2.2

整体算法 lightweight-resugaring 定义如下

算法 Lightweight-resugaring：给定 Surfexp 的表达式 Exp，输出其重组糖序列

```
Lightweight − resugaring(Exp)
    while(tmpexp==f(Exp)))
        if(tmpexp is empty):
            return;
        else if(tmpexp is surfexp or commonexp):
            output tmpexp;
            Lightweight − resugaring(tmpexp);
        else:
            Lightweight − resugaring(tmpexp)
```

3. 正确性证明

首先，由于我们的算法和先解糖后重组糖的区别就是我们只在需要的时候将语法糖解开，而传统意义上语法糖是先将语法糖全部展开然后再 CoreLang 上进行执行。

其次，为了证明方便，定义一些术语。

$(Headid\ Subexp_1\ Subexp_{...}\ldots)$ 为任意可规约表达式

对表达式运用 Headid 的对应规则进行规约，称为外部规约。

对其子表达式 $Subexp_i$ 规约。其中 $Subexp_i$ 为 $(Headid_i Subexp_{i1} Subexp_{i...}\ldots)$

- 如果对 $Subexp_i$=$(Headid_i Subexp_{i1} Subexp_{i...}\ldots)$ 进行外部规约，则称为表面规约。
- 如果对 $Subexp_{ij}$ 进行规约，则称为内部规约。

例：

$(if\ \#t\ Exp1\ Exp2)$—>$Exp1$

外部规约

$(if\ (And\ \#t\ \#f)\ Exp1\ Exp2)$—>$(if\ (if\ \#t\ \#f\ \#f)\ Exp1\ Exp2)$

表面规约

$(if\ (And\ (And\ \#t\ \#t)\ \#t)\ \#f)\ Exp1\ Exp2)$—>$(if\ (And\ \#t\ \#t)\ Exp1\ Exp2)$

内部规约

对 Exp=$(Headid\ Subexp_1\ Subexp_{...}\ldots)$，$Exp$ 称为上层表达式，$Subexp_i$ 称为下层表达式。

3.1 仿真性

仿真性：求值序列需要和在 CoreLang 上的求值顺序相同，即存在 CoreLang 上的求值序列中的部分中间过程与该序列中的元素对应。

对核心算法 f 逐条进行分析。

首先 $Rule1.1$ 和 $Rule1.3$ 不会影响仿真性，因为其本身就是在对 CoreLang 的表达式进行操作；

对 $Rule1.2$ 是对子表达式进行规约（用核心算法 f），因为 $Coreexp$ 内部的上下文规则是定的，因此即使将表达式所有糖都进行展开，也是该位置先进行规约。而在递归调用 f 的过程中，由于表达式深度优先，如果对下层的表达式调用 f 满足仿真性，那么此处上层表达式也满足仿真性。

对 Rule2.1 和 Rule2.2.2，解糖操作也不会破坏仿真性。

对 Rule2.2.1，由于由于仿真性的定义，重组糖的序列是可以解糖到 CoreLang 上执行序列的子序列的，因此证明仿真性只需要证明我们的序列在 CoreLang 上的序列有对应。我们在对 $(Surfid\ Subexp_1\ Subexp_{...}\ ...)$ 进行正常运算时，也会将其依据 Surfid 的规约规则进行展开；而在我们的核心算法中，我们也是这样进行单步展开，区别在于内部的糖没有规约。由于 Rule2.2 对 Exp 进行外部规约后得到 DesugarExp'，并调用了 f(DesugarExp')

- 如果 DesugarExp' 是 $Coreexp$ 或 $Commonexp$ 或 $OtherCommonexp$，且其子表达式规约，因此对于将 Exp 完全解糖后也是该子表达式的位置进行规约，其他子表达式的内容不变，因此对 Exp 的该子表达式规约后完全解糖和对 Exp 整体先解糖再规约，得到的是相同的表达式。如下图1所示

- 如果 DesugarExp' 是 $Surfexp$ 或 $OtherSurfexp$，则此处将递归调用 f，由于表达式深度有限，如果下层的表达式满足仿真性，则此处也满足仿真性.

3.2 抽象性

抽象性：求值序列中只存在 SurfLang 中存在的术语，没有引入 CoreLang 中的术语。

抽象性的正确性是显然的，因为我们在每次输出都判断了输出的 Exp 是否是 $Surfexp$ 或 $Commonexp$。（在 lightweight-resugaring 算法中）

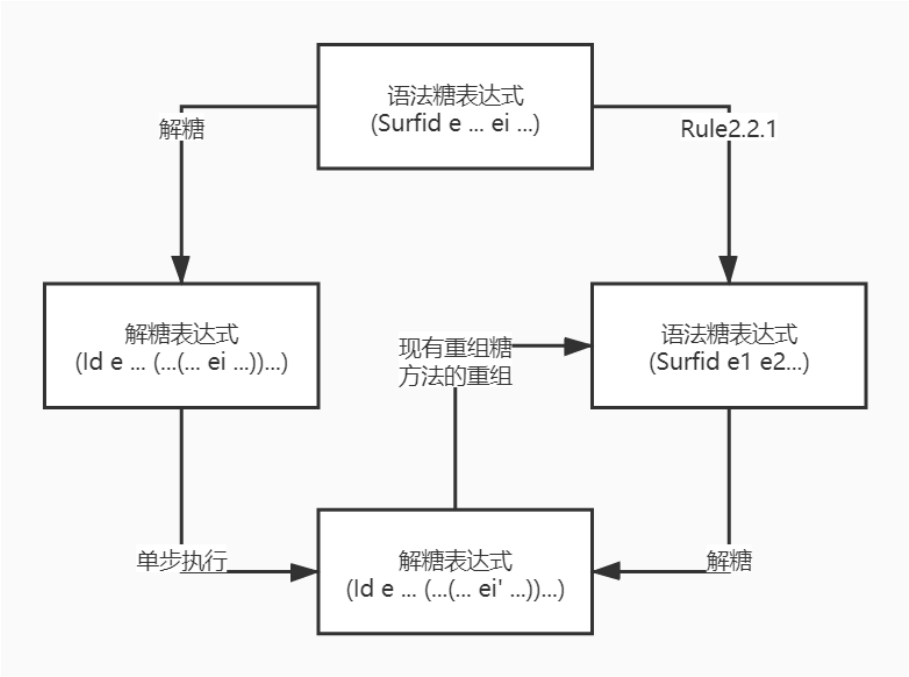

图 1 Rule2.2.1 仿真性图示

3.3　覆盖性

覆盖性：在求值序列中没有跳过一些中间过程。

引理：如果在重组糖序列中语法糖没有在不必须破坏的时候被破坏，那么就没有跳过中间过程。

引理证明：假设没有语法糖提前破坏的情况下，存在 CoreLang 序列中的某一项 Exp=$(Headid\ Subexp_1\ Subexp_{...}\ ...)$ 可被重组为 $ResugarExp'$=$(Surfid\ Subexp'_1\ Subexp'_{...}\ ...)$，且 todo

对 f 的每一条进行讨论。

对 Rule1.1 和 Rule1.3 显然没有破坏不必须破坏的语法糖。

对 Rule1.2，由于 CoreLang 的表达式需要进一步规约必须对特定子表达式进行规约，因此要对相应的下层表达式作用 f。同仿真性，由于表达式深度有限，下层表达式作用 f 满足覆盖性可递归证明上层表达式满足覆盖性。

对 Rule2.1，此时必须破坏语法糖，否则无法继续规约。

对 Rule2.2.1，与 Rule1.2 相同，递归的对下层表达式作用 f，递归证明覆盖性。

对 Rule2.2.2，因为单步尝试后发现语法糖解糖后，解糖后结构继续被破坏，因此无法还原回语法糖，也是必须破坏的语法糖。

因此求值序列没有破坏任何不必须破坏的语法糖，核心算法 f 满足覆盖性。

第四章 实现细节与评估

1. 实现细节

在 PLT Redex 中，对一个语言的定义主要需要三个内容。

- 文法
- 上下文环境
- 规约规则

我们在实现中，对所有表达式按如下文法进行子类划分。

```
exp ::= coreexp | surfexp | commonexp
```

与 (第三章第 1 节) 的定义有所差别的是，这里的 surfexp 包括了第三章第 1 节的 Surfexp 和 OtherSurfexp，commonexp 包括了第三章第一节的 Commonexp 和 OtherCommonexp。

对于 coreexp，自然是包含了所有内部语言的术语。因为我们的定义是将 Exp 的形式限制在 $(Headid\ Subexp_1 \ldots)$，通常情况下可以根据 Headid 判断是否是 coreexp。

例：

```
coreexp ::= (if exp exp exp)
|   (let ((x exp) ...) exp)
|   (first exp)
|   (rest exp)
|   ...
```

对于 surfexp，就是 $Headid$ 是表示 DSL 的标识的表达式。

例：

```
surfexp ::= (and exp exp)
|   (or exp exp)
|   (map exp exp)
|   (filter exp exp)
|   ...
```

对于 commonexp，是为了让我们的重组糖序列中有一些包含在内部语言、但是可以输出的中间过程。例如对于高阶糖

$$(Map\ f\ lst) --> (cons\ (f\ (first\ lst))\ (Map\ f\ (rest\ lst)))$$

我们需要一些中间序列用 cons 表示，来输出有用的中间过程。我们主要需要的 commonexp 有所有值、所有基础运算（包括算数运算和列表运算）。

对于上下文规则，对 coreLang 为了限制每个表达式只有一条规则可以规约，需要仔细限制位置。而对于 surfLang，由于我们不知道哪个子表达式需要先规约，要把每一个位置都设为可规约的洞。

```
(E  (v ... E e ...)
(let ((x v) ... (x E) (x e) ...) e)
(if E e e)
(and E e)
(and e E)
(or E e)
(or e E)
...
hole)
```

规约规则没有特殊说明。具体可参见代码。[1]

[1] https://github.com/yangdinglou/Lightweight-Resugaring-using-PLT-Redex

2. 评估

2.1 普通糖

我们首先就简单的

$(and\ e1\ e2) --> (if\ e1\ e2\ \#f)$

$(or\ e1\ e2) --> (if\ e1\ \#t\ e2)$ 糖进行测试。

因为我们的 CoreLang 上规定的求值顺序是，对 $(if\ e1\ e2\ e3)$ 的 $e1$ 求到值后就规约，因此我们应该看到的是对 and 和 or 糖，先将第一个值求到值，如何对糖进行破坏。

```
(and (and #t #f) (or #f #t))
(and #f (or #f #t))
#f
```

```
(and (or #f #t) (or #f #f))
(and #t (or #f #f))
(or #f #f)
#f
```

我们可以看到如上的两个例子皆按照我们预想的样子输出了重组糖序列。

另外尽管我们的重组糖序列不考虑有副作用的语言，但我们依然测试了一下考虑副作用时的糖的正确性。

$(Myor\ e1\ e2) --> (let\ ((tmp\ e1))\ (if\ tmp\ tmp\ e2))$ 其中没有用到任何其他糖，因此我们可以期待它的效果和上面的 or 糖一样。

```
(Myor (Myor #f #f) (and #t #t))
(Myor #f (and #t #t))
(and #t #t)
#t
```

结果显示我们的方法确实正确得到了该重组糖的序列。

2.2 复合糖

尽管和普通糖没有本质区别，但本节测试的糖在解糖后依然存在语法糖结构。我们构造如下 Sg 糖为例

$(Sg\ e1\ e2\ e3) --> (and\ (or\ e1\ e2)\ (not\ e3))$

```
(Sg (and #t #f) (not #f) #f)
(Sg #f (not #f) #f)
(and (or #f (not #f)) (not #f))
(and (not #f) (not #f))
(and #t (not #f))
(not #f)
#t
```

通过简单的展开验证我们发现，对于该语法糖表达式我们的方法确实做到了在语法糖需要展开的时候展开。

2.3 卫生宏

卫生宏定义：展开宏时变量作用域不被改变的宏。

例：

对语法糖 $(Let\ x\ v\ exp)$–>$(Apply\ (\lambda\ (x)\ exp)\ v)$

表达式 $(Let\ x\ 1\ (Let\ x\ 2\ (+\ x\ 1)))$ 中的 (+ x 1) 将用值为 2 的 x，因为他在内部 Let 的作用域内。

实现卫生宏的方法有很多，但对于现有的重组糖方法，卫生宏的重组糖[3]并不是那么容易。而在我们的方法中，实现卫生宏的重组糖很简单。

以表达式 $(Let\ \ x\ \ 1\ \ (Let\ \ x\ \ 2\ \ (+\ \ x\ \ 1)))$ 为例。我们会先将它单步展开到 $(Apply\ (\lambda\ (x)\ (Let\ x\ 2\ (+\ x\ 1)))\ 1)$，接着用 Apply 的规则进行规约，发现将 $(Let\ x\ 2\ (+\ x\ 1)))$ 中的 x 置换为 1 后，表达式并不符合规则，因此这不是一条有效的规约，不能对外层 Let 糖先展开。因此输出的第一个重组糖中间序列是 $(Let\ x\ 1\ (+\ 2\ 1))$

而在实现过程中，我们发现借助 PLT Redex 的 $\#:refers-to$ 命令，我们可以更简单的实现更漂亮的卫生宏重组糖。

测试结果如下。

```
(Let x (+ 1 2) (+ x (Let x (+ 1 4) (Let x 3 (+ x (Let x (+ 1 4) (+ x 1))))
((λ (x) (+ x (Let x (+ 1 4) (+ x 1)))) 3)
(+ 3 (Let x (+ 1 4) (+ x 1)))
(+ 3 (Let x 5 (+ x 1)))
(+ 3 ((λ (x) (+ x 1)) 5))
(+ 3 (+ 5 1))
(+ 3 6)
9
```

2.4 递归糖

递归糖是对于现有重组糖方法很难处理的一种语法糖，指的是多个语法糖相互调用。在本节我们主要用如下例子说明。

$(Odd\ e)$–>$(if\ (>\ e\ 0)\ (Even(-\ e\ 1)\ \#f))$

$(Even\ e)$–>$(if\ (>\ e\ 0)\ (Odd(-\ e\ 1)\ \#t))$

看似不复杂的两条语法糖规则，对于现有方法来说并不是一件容易的事情。具体原因我们将在下一高阶糖和下一节中仔细探讨。测试结果如下：

```
(Odd 6)
(Even (- 6 1))
(Even 5)
(Odd (- 5 1))
(Odd 4)
(Even (- 4 1))
(Even 3)
(Odd (- 3 1))
(Odd 2)
(Even (- 2 1))
(Even 1)
(Odd (- 1 1))
(Odd 0)
#f
```

2.5 高阶糖

高阶糖本质上和递归糖没有区别，

我们分别用两种不同的语法糖形式实现了 map 和 $filter$ 两个语法糖。

$(map\ e\ (list\ v_1 \ldots))$–>

$(if\ (empty\ (list\ v_1 \ldots))\ (list)\ (cons\ (e\ (first\ (list\ v_1 \ldots)))\ (map\ e\ (rest\ (list\ v_1 \ldots)))))$

$(filter\ e\ (list\ v_1\ v_2 \ldots))$–>

$(if\ (e\ v_1)\ (cons\ v_1\ (filter\ e\ (list\ v_2 \ldots)))\ (filter\ e\ (list\ v_2 \ldots)))$

$(filter\ e\ (list))$–>$(list)$

可以看出，对 map 糖我们将边界条件用解糖后的 if 表达式进行了限制；而对 filter 糖，我们用两个糖的参数区分了不同的条件用来表示边界条件。在保证同名不同参数的糖没有二义性的前提下，我们允许一糖多参，让书写语法糖更加简单。

测试结果如下

```
(map (λ (x) (+ 1 x)) (list 1 2 3))
(cons 2 (map (λ (x) (+ 1 x)) (list 2 3)))
(cons 2 (cons 3 (map (λ (x) (+ 1 x)) (list 3))))
(cons 2 (cons 3 (cons 4 (map (λ (x) (+ 1 x)) (list)))))
(cons 2 (cons 3 (cons 4 (list))))
(cons 2 (cons 3 (list 4)))
(cons 2 (list 3 4))
(list 2 3 4)
```

```
(filter (λ (x) (and (> x 3) (< x 6))) (list 1 2 3 4 5 6 7))
(filter (λ (x) (and (> x 3) (< x 6))) (list 2 3 4 5 6 7))
(filter (λ (x) (and (> x 3) (< x 6))) (list 3 4 5 6 7))
(filter (λ (x) (and (> x 3) (< x 6))) (list 4 5 6 7))
(cons 4 (filter (λ (x) (and (> x 3) (< x 6))) (list 5 6 7)))
(cons 4 (cons 5 (filter (λ (x) (and (> x 3) (< x 6))) (list 6 7))))
(cons 4 (cons 5 (filter (λ (x) (and (> x 3) (< x 6))) (list 7))))
(cons 4 (cons 5 (filter (λ (x) (and (> x 3) (< x 6))) (list))))
(cons 4 (cons 5 (list)))
(cons 4 (list 5))
(list 4 5)
```

以 $(map\ (\lambda\ (x)\ (+\ 1\ x))\ (list\ 1\ 2\ 3))$ 为例。如果将该糖完全展开，将变成如下表达式

$$
\begin{array}{l}
(if\ (empty\ (list\ 1\ 2\ 3)) \\
\quad (list) \\
\quad (cons\ ((\lambda\ (x)\ (+\ 1\ x))\ (first\ (list\ 1\ 2\ 3))) \\
\qquad (if\ (empty\ (list\ 1\ 2\ 3)) \\
\qquad\quad (list) \\
\qquad\quad (cons\ ((\lambda\ (x)\ (+\ 1\ x))\ (first\ (list\ 1\ 2\ 3))) \\
\qquad\qquad (if\ (empty\ (list\ 1\ 2\ 3)) \\
\qquad\qquad\quad (list) \\
\qquad\qquad\quad (cons\ ((\lambda\ (x)\ (+\ 1\ x))\ (first\ (list\ 1\ 2\ 3))) \\
\qquad\qquad\quad (map\ (\lambda\ (x)\ (+\ 1\ x))\ (rest\ (list\ 1\ 2\ 3)))))
\end{array}
$$

todo:SKI with lazy lambda.

3. 与现有方法对比

正如前文多次提到的，我们的方法与现有重组糖方法最大的不同是，不会完全将语法糖展开。这个区别导致了我们需要限制语法糖和内部语言的结构化1。该限制最大的影响就是语言不能有副作用，我们将在下一节对该问题进行讨论。抛去这点劣势，我们的重组糖相比现有方法有如下的优点。

- 轻量级。不将语法糖完全展开，意味着不需要对语法糖表达式展开后的内部语言表达式的多步执行过程进行一次又一次的匹配和替代。因为对于一个相对比较大的程序来说，这样的匹配开销并不小，因此我们的做法—只在需要（不得不）展开语法糖时再将语法糖破坏的方法，理论上很大程度节省了重组糖的时间开销。

- 对卫生宏友好。重组糖系列工作第二篇用一个新的数据结构—抽象语法有向无环图，来处理卫生宏的重组糖问题；而正如上一小节所介绍2.2，我们的系统处理卫生宏和普通宏的区别只在于借用 PLT Redex 的重命名；并且即使不用这个功能，算法依然可以有效运行。[2]这条优点本身也属于一种“轻量级”的表现—算法本身简单以至于拓展性强，而现有工作则对算法进行重新设计。

- 更多语法糖特性（尤其递归语法糖）。正如前一节所介绍的，我们的方法处理递归糖和高阶糖的能力，是现有方法很难达到的。原因很简单：递归糖需要处理终止条件，而本身语法糖处理终止条件就不是一件容易的事情。而得益于规约语义的强大能力，加上算法本身的精心设计，让处理递归这件事成为可能，进而让高阶语法糖成为可能。我们可以发现，高阶函数作为函数式语言的一个非常重要的特性，在近年来被多种其他编程范式的语言所支持。因此支持递归的语法糖是本工作十分重要的优势。

- 更简单的书写语法糖。相比于 resugaring 系列工作 [4]，我们的方法得益于规约语义的友好形式，对语法糖的书写并不必须是模式匹配为基础的。只要可以保证对同一个语法糖没有出现多种解糖方式，我们可以对语法糖的各种条件进行分类处理。比如说实现一个斐波那契数列语法糖，我们可以设计如下解糖规则。（对应写在规约规则中）

  $(f\ 0)$–>1

  $(f\ 1)$–>1

  $(f\ x)$–>$(+\ (f\ (-\ x\ 1))\ (f\ (-\ x\ 2)))$ side-condition (x>1)

  这种语法糖写法相对于简单的 pattern based 语法糖，更加友好。

[2]需要对 Rule2.2 进行微小改动，代码中也有运行实例

4. 一些其他讨论

4.1 副作用

表达式不能用对外副作用一定程度我们我们方法的便利性。在一定解决范围内，我可以基于 let（变量绑定）解决部分需求。

不能有副作用的主要原因，是因为：一旦语法糖的子表达式带有副作用，而当表达式并没有执行到带副作用语句前是可以重组糖的，执行后便不能重组。而对于嵌套表达式来说，判断是否执行过有副作用的语句是一件很困难的事情。可能的解决方案是设计一个检测副作用的算法来增强我们的系统。

4.2 其他相关工作

第五章 总结与展望

1. 结论

我们的重组糖方法相对于现有方法，是一种更轻量级的算法—它以基于归纳语义的 PLT Redex 为基础，用单步尝试加上不完全展开的基础思想，解决了现有工作不能处理递归糖、高阶糖的问题，并且让处理语法糖尤其是卫生宏的流程简单而自然。

2. 未来可能的工作

2.1 改进本方法，支持副作用语法糖

2.2 DSL 解释器

我们的工作初衷是为了解决如下的问题：

> 给定内部（通用语言）的解释器（或求值规则），加上外部语言（领域特定语言）的映射关系，用算法自动推导出外部语言（不依赖内部语言）的解释器（或求值规则）。

在对问题初步思考后，结合学习阅读 resugaring 系列工作，提出了本文主要工作的想法。而我们的最终目标依然没有变。

广义上讲，本文实现的工具是一个解释器—它将带有语法糖的表达式，在表面语言上一步一步的解释执行。但最终这个解释执行依然是包含算法对内部语言的一些尝试，没有脱离对内部语言的依赖。然而本工作核心算法的部分思想对此目标有借鉴意义，今后的工作可能会朝这方向发展，将该方法抽象到符号层面。

2.3 多步程序综合

参考文献

[1] 期刊

[2] 网络文档

1. 期刊作者. 论文名. 刊名, 出版年份, 卷号 (期号): 起始-截止页

2. 专著作者. 书名. 版本 (第一版不写). 出版城市: 出版社, 出版年份: 起始-截止页

3. 论文集论文作者. 论文题目//编者. 论文集名: 其他题名信息. 出版城市 (或者会议城市): 出版者, 出版年: 引文起始-截止页码

4. 学位论文作者. 学位论文题名. 城市: 论文保存单位, 年份

5. 网络文献作者. 题名 [文献类型标志/文献载体标志]. 出版地: 出版者, 出版年 (更新日期)[引用日期]. 获取和访问路径

* 注意: 作者姓前名后, 超过 3 名作者列前 3 名, 后加“, 等”; 英文姓名, 姓前名后, 姓首字母大写, 名缩写; 文献的项目要完整, 各项的顺序和标点要和格式要求一致; 未公开发表的论文、报告不列入正式文献, 如有必要可在正文当页下加注。英文文献格式同上。参考文献在正文中按出现顺序用 [1], [2]..... 在右上角标注, 放在“参考文献”中时, 用 [1], [2], ... 顺序标注。

参考文献

[1] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780, May 1935.

[2] Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 1st edition, 2009.

[3] Justin Pombrio and Shriram Krishnamurthi. Hygienic resugaring of compositional desugaring. *SIGPLAN Not.*, 50(9):75–87, August 2015.

[4] Justin Pombrio and Shriram Krishnamurthi. Resugaring: Lifting evaluation sequences through syntactic sugar. *SIGPLAN Not.*, 49(6):361–371, June 2014.

这是参考“参考文献”，主要用来看引用的顺序，请手动些参考文献或自行写程序，最终编译请删除

本科期间的主要工作和成果

本科期间参加的主要科研项目
本研基金

1. 基金名称. 基金类型. 指导老师. 基金支持年限

各种科研项目

1. 项目名称. 项目类型

格式下
期刊：
全部作者. 论文名. 期刊名, 出版年份, 卷号 (期号): 起始-截止页
会议论文：
全部作者. 论文名. 会议名, 会议举办地, 会议举办时间, 起始-截止页
专利
全部专利申请人. 专利名称. 专利申请号. 专利申请日期. 国别

致谢