

An Interactive System For Terms And Substitutions

(Master's Project)

Fengzhang Liu

Advisor: Paliath Narendran

Abstract

A very important research field, term rewriting is widely used in many areas, such as cryptographic protocol analysis and automated theorem proving. The goal of this project is an interactive system for terms and substitutions to help beginners with some key concepts. It is implemented in the programming language Python.

Acknowledgements

I would like to express my thanks to the people who offered help to me during the project. I am grateful to my advisor Prof. Paliath Narendran, who gave nonstop support. In particular, I want to thank the classmates who assisted me to complete the project by exchanging their own interesting ideas and thoughts, as well as providing accurate information. Besides, I am also very thankful to my parents because they always support me and inspired me to go my own way. Last but not the least, I want to thank my friends who encouraged me to do hard work.

1 Introduction and Motivation

As an important part of automated reasoning, term rewriting system derives from mathematical logic. The system consists of sequences of discrete transformation steps in which one term is replaced by another equivalent one. This areas has many applications, such as functional programming, automatic proving of theorems and formal verification.

This interactive system for terms and substitutions is designed with the purpose of helping students to understand basic concepts involved in term rewriting. It can be considered as a calculator on terms. Terms and substitutions (input the specific term and apply a substitution thereby getting a new term) are taken into account. So in this system, five functions are carried out. An introduction to these concepts is given below:

Substitution: A substitution is a mapping from variables to terms. It can be naturally extended to a mapping from terms to terms. A term t will be a substitution instance of a term s if and only if t is obtained from s by substituting terms for variables in s , replacing each occurrence of the same variable by an occurrence of the same term.

Composition: Let Σ be a signature and V be a countably infinite set of variables. Composition $\sigma\tau$ of two substitution σ and τ is defined as follows: $\sigma\tau(x) := \hat{\sigma}(\tau(x))$ for all variables x . $\sigma\tau$ is a mapping from V into $T(\Sigma, V)$.

Unification: The unification problem can be expressed as: given two terms containing variables find a most general substitution, if one exists, which makes the two terms equal. The resulting substitution is called a most general unifier.

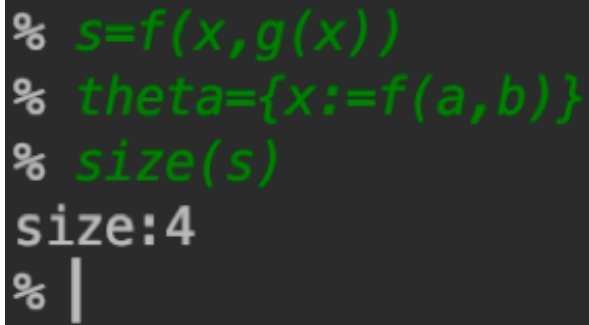
2 Terms and Substitutions System

The TERMS AND SUBSTITUTIONS SYSTEM is used for manipulating symbolic term expressions. The currently implemented functionalities are *substitution*, *composition*, *unification*, *size* and *height* (of terms).

We follow the definitions and notations from the book *Term Rewriting and All That* [1].

The source code is implemented in using Python 3. So as long as the operating system compiles Python 3, the program will work on Windows, Linux and MacOS.

2.1 System interface and entering commands

A screenshot of a terminal window with a dark background. It shows a series of commands entered in green text: `% s=f(x,g(x))`, `% theta={x:=f(a,b)}`, and `% size(s)`. The output `size:4` is shown in white text. A small white vertical bar, representing a cursor, is positioned at the end of the last command line.

```
% s=f(x,g(x))
% theta={x:=f(a,b)}
% size(s)
size:4
% |
```

The above figure is the interface of the project. The cursor is the small vertical bar that blinks in the text window. The text which is currently being written in the text window and can be submitted to the system for evaluation is rendered in green. The text that the system sends back is rendered in white.

2.1.1 Defining terms and substitutions

To use the system, the user will need to define terms and substitutions. Terms can be defined by using a letter or letter with numbers. In addition, substitutions can be defined by using the name *theta* or theta with a number. An example of defining terms and substitutions is shown below:

```
%s = f(x , g(x))
%theta = {x:=f(a, b)}
%theta1 = {x:=f(y, y)}
%s1 = g(x, f(x, y))
```

2.1.2 Application of a substitution

A specific term will be input and a substitution will be applied, in order to get a new term. If two terms or substitutions are applied, the system will return “Wrong Input”. An example of applying a substitution to a term is shown below:

```
%theta = {x:=f(a, b)}
%t = f(x, g(x))
%apply(theta, t)
f(f(a, b),g(f(a, b)))
```

2.1.3 Composition of two substitutions

The compose function composes two substitutions. If two terms or one term and another substitution are input, the system will return “Wrong Input”. An example of composing two substitutions is shown below:

```
%theta1 = {x:= f(y, y)}  
%theta2 = {y:= g(a)}  
%compose (theta1, theta2)  
{x:=f(y, y), y:=g(a)}  
%compose (theta2, theta1)  
{x:= f(g(a), g(a)), y:=g(a)}
```

2.1.4 Unification

The unification will process two terms and then find a *most general* substitution that unifies them. In terms of this function, command line “unify” is used for unifying two terms. As for error check, there are two situations, including “function clash” and “occur check”. Then the input of unification function must have two terms. If the input refers to two substitutions or one term and another substitution, the system will return “Wrong Input”. An example of unification is shown as below:

```
%s = p(a, Y)  
%t = p(X, f(b))  
%unify (s, t)  
{X:=a, Y:=f(b)}  
%s1 = f(X, g(X))  
%t1 = f(Y, Y)  
%unify(s1, t1)  
occur check  
%s2 = p(f(X, Y), Z)  
%t2 = p(g(Y), b)  
%unify(s2, t2)  
function clash
```

2.1.5 Size and height of terms

The *size* and *height* functions can be applied to find the size and height of terms respectively. Since only the size and height for terms can be found, the system will return “Wrong Input” if size and height with substitutions are applied.

```
%s = g(x, f(x, y))
%size(s)
size 5
%height(s)
height 2
```

The size of a term is the number of nodes in the tree representation of the term and the height of a term the length of the longest path from the root of the tree to a leaf node.

3 Conclusion and Future Work

This project focuses on developing a study tool for term rewriting. The purpose of this system is to help students better understand term rewriting. The implemented basic functions are *substitution*, *composition*, *unification*, *size* and *height*. The implementation is in the programming language Python. In the future, it will be necessary to consider additional functions, such as E-unification and Homeomorphic Embedding. As more features are added, the system will also become more useful.

References

- [1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.
- [2] M. Bezem, J. Klop, and R. Vrijer. *Term Rewriting Systems* Cambridge, UK: Cambridge University Press. 2003.
- [3] C. Bouchard, K.A. Gero, C. Lynch, and P. Narendran. On Forward Closure and the Finite Variant Property. *Frontiers of Combining Systems*. Lecture Notes in Computer Science. Springer, 2013.
- [4] E. Ohlebusch. *Advanced topics in term rewriting*. New York: Springer, 2011.

A Substitution

```
% theta1 = {X:= f(a,b)}
% t = f(a,g(X))
% apply (theta1, t)
f(a, g(f(a, b)))
% theta2 = {X:= a, Y:= f(b, b)}
% t2 = f(f(X, b),g(Y, a))
% apply (theta2, t2)
f(f(a, b),g(f(b, b), a))
% theta3 = {X:= a, Y:= b}
% t3 = f(X ,g(X, f(Y)))
% apply (theta3, t3)
f(a, g(a, f(b)))
% theta4 = {X:= g(a, f(b, a)), Y:= f(g(a), a)}
% t4 = f(X, f(X, Y))
% apply(theta4,t4)
f(g(a, f(b, a)), f(g(a, f(b, a)), f(g(a), a)))
% theta5 = {X:= h(X)}
% t5 = f(X,X)
% theta6 = {X:= h(a)}
% apply(theta5, t5)
f(h(X), h(X))
% apply(theta6, t5)
f(h(a), h(a))
```

B Composition

```
% theta1 = {X:= f(Y, Y)}
% theta2 = {Y:= g(a)}
% compose(theta1, theta2)
{Y:= g(a),X:= f(Y,Y)}
% compose (theta2, theta1)
{X:= f(g(a), g(a)), Y:=g(a)}
% compose(theta1, theta1)
{X:=f (Y,Y)}
% compose(theta2, theta2)
{Y:= g(a)}
% theta3 = {X:= h(X)}
% theta4 = {X:= h(a)}
% compose(theta3, theta3)
```

```

{X:= h(h(X))}
% compose(theta4, theta3)
{X:= h(h(a))}
% theta5 = {X:= Y, Y:= X}
% compose(theta5, theta5)
{}

```

C Unification

```

% s1 = h(f(X), Y)
% t1 = h(Y, f(Z))
% unify(s1, t1)
{Y:= f(X), Z:= X}
% s2 = f(a, X)
% t2 = f(Y, b)
% unify(s2, t2)
{Y:= a, X:= b}
% t3 = f(f(Y, Z), f(Z, b))
% s3 = f(X, X)
% unify(s3, t3)
{Z:= b, Y:= b, X:=f(b, b)}
% s4 = f(f(X, Y), f(Z, W))
% t4 = f(f(Y, Z), f(W, X))
% unify(s4, t4)
{Z:= Y, W:= Y, X:= Y}
% s5 = f(f(X, Y), f(U, V))
% t5 = f(f(Y, X), f(U, V))
% unify(s5,t5)
{X:=Y}
% s6 = f(X, X)
% t6 = f(a, b)
% unify(s6, t6)
Function Clash
% s7 = f(X, f(g(X), b))
% t7 = f(a, f(Y, Y))
% unify(s7, t7)
Function Clash
% s8 = f(g(X), f(X, b))
% t8 = f(c, f(g(X), X))
% unify(s8, t8)
Occur-Check

```