单位代码 10006
学　　号 10091204
分 类 号 TP301.2
密　　级

北京航空航天大學
BEIHANG UNIVERSITY

毕业设计(论文)

不确定性的、带有附带影响的子表达式的统一程序理论框架下的新的语义

院（系）名称　数学与系统科学学院
专 业 名 称　华罗庚实验班计算机与信息数学
学 生 姓 名　齐嘉悦
指 导 教 师　漆毅

2014 年 6 月

北京航空航天大学

本科生毕业设计（论文）任务书

Ⅰ、毕业设计（论文）题目：

不确定性的、带有附带影响的子表达式的统一程序理论框架下的新的语义

Ⅱ、毕业设计（论文）使用的原始资料（数据）及设计技术要求：

UTP 手册【HH98】（the UTP book）,其中的主要章节是：

•关系（Relations）第二章，2.9 节，这部分主要关注的问题是局部变量。

• 设计（designs）第三章。

• 更高阶的程序（Higher Order Programming）第九章，9.3 节，这部分主要讲的是函数。

• 链接理论（Linking Theories）第四章，这部分涵盖了链接理论的进一步讨论并且有着最简要的关于 UTP 的展示介绍。

Ⅲ、毕业设计（论文）工作内容：

1. 使用 UTP 的框架[HH98]来对于拥有全局变量的、附带影响以及任意子表达式的重新排序的命令式语言给出语义；
2. 探索怎样形式化地定义“安全”(“不安全”仅仅就是因为缺乏确定性吗？)；
3. 探索怎样描述安全的子集的特征（像 MISRA-C 这样的)；
4. 将在方案中研究所给出的语义和确定性无附带影响的语言的关系。

Ⅳ、主要参考资料：

[CC+00]C and c++ coding standards. bssc(2000)1 issue, 2000.

[Mis04]MISRA Consortium. MISRA-C:2004—Guidelines for the use of the C language in critical systems, 2004.

[HH98]C.A.R Hoare and Jifeng He. Unifying Theories of Programming. Prentice-Hall, 1998.

数学与系统科学学院 学院（系） 计算机与信息数学 专业

类 华罗庚实验 班

学生 齐嘉悦

毕业设计（论文）时间： 2014 年 3 月 4 日至 2014 年 6 月 3 日

答辩时间：　　年　　月　　日

成　　绩：

指导教师：

兼职教师或答疑教师（并指出所负责部分）：

系（教研室） 主任（签字）：

注：任务书应该附在已完成的毕业设计（论文）的首页。

本人声明

我声明，本论文及其研究工作是由本人在导师指导下独立完成的，在完成论文时所利用的一切资料均已在参考文献中列出。

作者：齐嘉悦

签字：

时间：2011 年 6 月

不确定性的、带有附带影响的子表达式的统一程序理论框架下的新的语义

学生：齐嘉悦

指导教师：Andrew Butterfield

指导教师：漆毅

摘要

为了改善 C 语言要求太过宽松、语法规范不够严格等给具体程序带来的问题，我们将利用程序的统一理论来为 C 语言或者类似于 C 语言的不确定性的、带有附带影响的这些子表达式赋予新的语义，减少程序员在具体操作过程中的麻烦和不方便之处。我们的主要技术路线是首先探索出一套新的语义，然后一一探究它们是否符合规定的健康条件；之后根据证明过程对于先前的语义规定做出改进。文章在具体的检验新的语义之前还对于十一条基本的法则进行了健康性的检验证明，后面开始对十条语义定义的检验证明；具体的证明方法主要来自于形式化方法这门课程，有很多一阶谓词演算的基本思想。最终的结论中我们成功给出了一套语法和语义定义，尽管未来还有很大的完善空间。

关键词： C 语言语义，程序的统一理论，不确定性，附带影响

UTP semantics of non-deterministic side-effecting expression

Author: Jiayue Qi
Tutor: Andrew Butterfield
Tutor: Yi Qi

Abstract

C programming language is safety-critical, in order to reduce the problems caused by its loose semantics, we are going to use Unifying Theories of Programming framework to give a semantics to an imperative language with global variables, side-effecting functions, and arbitrary sub-expression reordering, and then use this to explore how to formally define "safety". The main technology roadmap we use in this paper is to first find out a set of new semantics and then check the healthiness of eleven basic laws, then use those laws to check the healthiness of our new semantics to see if they match the prescript healthiness conditions，then we do some work to make the previous semantics better. The most important ways of doing proofs are from the module Formal Methods and many ideas related to First-order predicate calculus. At last, we gained a new semantics and definitions of syntax successfully though there is still a long way to go.

Key words: C semantics, Unifying Theories of Programming, non-deterministic, side-effecting

目录

1 绪论.....1
2 建议使用的类似于 C 语言的语言.....4
3 类似于 C 语言的语言的语义.....6
3.1 语法.....6
3.1.1 细节说明.....6
3.1.2 健康性.....7
3.2 语义.....8
4 基本法则.....10
5 阶乘的例子.....18
5.1 对数阶乘.....18
5.2 依赖于历史行为的附带影响.....19
6 语义的健康条件检验.....21
6.1 k 是否健康?.....21
6.2 v 是否健康？.....21
6.3 $e1 \odot e2$ 是否健康？.....21
6.4 $\mathrm{Re}\,t(e)$ 是否健康？.....23
6.5 $f(e)$ 是否健康？.....24
6.6 $v := e$ 是否健康？.....25
6. 7 $Skip$ 是否健康？.....25
6. 8 $p;q$ 是否健康？.....25
6.9 $p \lhd c \rhd q$ 是否健康？.....28
6.10 $c \circledast p$ 是否健康？.....28
结论.....29
致谢.....30
参考文献.....33

1 绪论

“C 语言是诡诈的、有缺陷的，但也是一个巨大的成功。”C 语言之父丹尼斯·里奇如是说道。

诚然，C 语言是一个巨大的成功，我们可以就这点分以下几个小点来陈述：[1]首先，C 语言非常简洁和灵活。相较而言，C 语言有更小量的关键词——ANSI C 标准只有 32 个关键词和 9 个控制语句，这显然是削减掉了那些不必要的部分；第二点，C 有很强的便捷性，不同机器上的 C 语言有 80% 都是可共享的；第三，它有一个相对强大的表达能力：它有很多数据类型，如整型，字符串，实型，数组，结构，联合类型，枚举类型，指针等供程序员来选择；第四，我们可以用它来实现一个结构化的程序设计：C 语言将函数看作程序设计的基本单元，函数在C语言中的作用就像子程序在类似语言中的作用；另外，它会产生高质量的程序，而且它可以在电脑的硬件中被操作。

然而，“C 是有缺陷的” ，这是因为它有的时候太过“灵活”，对于电脑的控制有时候太过自由使得有时就会出现一些错误：C 语言的主要缺陷就在于数据的封装性，[2]这一点会导致数据安全性方面的一个很大的缺点，这也是 C 和 C++之间的主要区别。另外，C 有一个相对自由的语法规则和宽松的对于变量类型的限制，有些时候这就会导致很多不好解决的问题。

举例来说，在 C 语言的陈述中：

```
v=f(a)+g(b)
```

我们并不能保证 f 就会在 g 之前被赋值，v 也不一定就是值变化了的唯一全局变量。由于 f 和 g 的附带影响可能与赋值顺序有关, 当我们考虑到程序的安全性时，调换它们的执行顺序就并不是一件好事。根据 MISRA-C[3]（一个细化的限制子集，用来保证展示让监管和鉴定权威机构都满意的程序结果的可行性），这段程序就应该被写成这样：

```
t1=f(a);
t2=g(b);
v=t1+t2
```

这里 t1 和 t2 都是新鲜的变量。

下面让我们来看另一个例子：

x+x=2*x

这是一个众所周知的基本代数法则，现在让我们来看一个函数：

logfac:

```
int logfac(int n) {
int n,f;
x=n;
f=1;
while (x $>$ 1) {
f=x*f;
 x=x-1
}
count =count+1;
return (f)
```

接下来，我们给出如下所示的两个程序：

y1 = logfac(z)+ *logfac(z)*

y2 = 2* *logfac(z)*

现在我们发现上面说到的那个众所周知的基本代数法则竟然在这里不能使用了！那是因为在程序 y1 中，count 被加了两次，但是在程序 y2 中，count 只被加运算操作了一次！

这个例子就说明了附带影响的影响以及它们如何阻止我们使用基本的代数规则，即使这些规则看起来是可以使用的。

因此这些例子在某种程度上告诉了我们 C 程序语言的缺陷。

为了对 C 程序语言做出一些贡献，我们决定使用程序语言的统一理论（UTP）框架[HH98]来对于拥有全局变量、附带影响以及任意子表达式的重新排序的命令式语言给出语义，然后用这一套语义来探索怎样形式化地定义``安全性''以及怎样找到安全子集的特征，并探索它们和一个确定性的、无附带影响的语言的关系。

程序的统一理论谈论[4]程序的语义并且告诉我们操作性语义、代数语义和指称语义是怎样在一个统一框架中结合起来的，以此来满足程序和计算机系统的设计、贯彻和形式化细化。一阶谓词演算就是程序的统一理论的语义基础。根据艾瑞克·海涅的理论说

法，在程序的统一理论中程序就是命题；在语义层面程序和说明是没有区别的。在统一程序理论中，一个理论就是特定程序样式的一个模型，一个统一程序理论就是由那些基本元素组成的：一个字母表，一个签名和一个健康条件的集合。精炼也是一个统一程序理论中的重要概念。程序的精炼是从详述开始的通过应用那些保留了正确性的规则的一个结构。

现在就来到了本论文的基本结构和内容部分。这篇论文的主要目的就是通过使用统一程序理论的框架来对一个命令式的、拥有全局变量的和附带影响函数的、任意子表达式可重新排序的语言赋予一套新的语义；另外一个目的就是探索怎样应用 $U\bullet(TP)^2$ 定理证明器来辅助我们达成第一个目标。我们基本的研究步骤将像下面所说的：

（1） 探索出一套合理的语义；

（2） 利用 $U\bullet(TP)^2$ 的辅助完成十一条基本法则的证明；

（3）利用基本法则等辅助研究语言的语义是否符合健康条件：如果是，给出证明；如果不是，说明语义或者健康条件的正确性；

（4）完善语义规定。

创新点：将一个 C 语言（不确定性表达）语义中的复杂部分形式化，而不是坚持使用开始阶段的语言的简单子集合。

难点一： 定义所有的健康条件；

难点二： 怎样抓住程序子集合的不确定性特点。

2 建议使用的类似于 C 语言的语言

我们用一个简单的语言举例并将它扩展为如下:[5]

（1） 我们添加了一个用来定义和称呼带有参数的函数的能力;

（2）我们给表达式添加了一个具体的语法，它们现在被看作具有附带影响的状态;

（3）我们规定子表达式被赋值的顺序是被非确定性地选择出来的。

我们假定函数的定义是静止的，并且是独立于程序主体被给出的，它们形成了程序执行的一个背景：

$$p \vdash q = ok \wedge p \Rightarrow ok' \wedge q \quad f(v,\cdots,v) \triangleq p \in FuncDefn$$

这里 p 是一个扩展的程序语法中一个普通的程序,现在我们定义一个表达式的语法，被函数化地称作可能性：

$k \in Constant$ 文字或者常量

$\odot \in Operator$ 二元运算符, 如： $+,-,\times,/,\leq,=,\ldots$

$e \in Expr \quad ::= \quad k \mid v \mid e \odot e \mid f(e,\ldots e)$

然后我们来看我们的程序语法，并将它扩展使它能够包括一个详细的返回状态:

$$p \in \mathrm{Pr}og ::= Skip \mid v := e \mid p;q \mid p \lhd c \rhd q \mid c \circledast p \mid \mathrm{Re}t(e)$$

下面让我们来逐步探索所给语言的语义：

$p,q \in \mathrm{Pr}og ::= skip$ 不进行任何操作 c \circledast p

| $v := e$ 赋值

| $p;q$ 执行 p, 然后执行 q

| $p \lhd c \rhd q$ 如果 c 成立就执行 p, 否则执行 q

| $c \circledast p$ 当 c 成立时执行 p

说明：

（1）skip 什么都不做，但是将它放在此处是很有用的;

（2）赋值语句中我们用的是 :=（变成）而不是 = ，这是为了避免与数学中的等于产生混淆;

（3） “;” 是一个将程序联系起来的操作符；

（4）现在，我们暂时忽略变量的声明。

3 类似于 C 语言的语言的语义

为了使得建议的程序的统一理论更加整齐，我们将把列出的表达式扩展为充实的陈述。[6]

举例来说，表达式 x:=e 有以下的意义：

我们为 e 赋值这件事也许会导致一些附带影响，我们使用 e 的返回值，并且用它来更新 v 的值。

如果我们有一个 能够评判 e 的“状态”，那么我们就可以根据它的附带影响给 e 赋值,然后我们可以直接忽略任何的返回值。

3.1 语法

当我们有了这个变化以后我们就可以按照以下方法来总结我们的非确定性、带有附带影响的语言的语法了：

$$f(v,\cdots,v) \triangleq p \in FuncDefn$$

$k \in Constant$ 文字或者常量

$\odot \in Operator$ 二元运算符，如：$+,-,\times,/,\leq,=,\ldots$

$$e \in Expr \quad ::= \quad k \mid v \mid e \odot e \mid f(e,\ldots e)$$

$$p \in \mathrm{Prog} ::= Skip \mid v := e \mid p;q \mid p \lhd c \rhd q \mid c \circledast p \mid \mathrm{Ret}(e)$$

3.1.1 细节说明

我们应该将 ok 看作一个模型变量，来注释稳定性，但是我们也应该加入另外一个特殊的变量 res 来注释 $Ret(\bullet)$的结果状态。

然而，我们应该将这个看作一个初步的、并非一个模型的变量，因为它使我们能够按照一种令语义定义更加模式化的方式来使用赋值语句。

我们假设一下两个变量都没有出现在我们的程序正文中：ok 和 res。

$$M = \{ok\}$$

$S = \{res\} \cup$program variables

3.1.2 健康性

现在我们来介绍“设计”的表示： $p \vdash q$ 。每一个设计 D 都可以被写成如下形式：

$$ok \wedge P \Rightarrow ok' \wedge Q$$

现在让我们来看看设计形式的简写：

《⊢-def》 $P \Rightarrow Q \triangleq ok \wedge P \Rightarrow ok' \wedge Q$

这个表示方式是不可用其他形式替代的。一个设计是一个命题被如此表达（或者可以被如此表达）的关系。

下面我要来介绍健康条件。

我们将设计看做是``健康的''命题，它们是用来描述真正的程序和规范的。“健康条件”就是那些用来测试一个命题是否健康的条件。这些条件捕捉了一个真正的程序的行为的关键特点，有一些是强制性的、所有程序都适用的，有一些是可选择的，它们描述行为表现良好的程序。我们一共有四个健康条件，但在这篇论文中我们只详细描述其中的两个（也就是强制性的）。

我们使用标准的设计的表示，采用关联的四个健康条件：

$$p \vdash q = ok \wedge p \Rightarrow ok' \wedge q$$

H1 $v := e ?= True \vdash \exists r_0 \bullet e[r_0 / res'][v' = r_0 \wedge S' \setminus v = S \setminus v \quad P = ok \Rightarrow P$

H2 $P = P;((ok \Rightarrow ok') \wedge S' = S)$

H3 $P = P; Skip$

H4 $P; true = true$

在 H1 条件中 P 只在 ok 是真的的时候 “有效”（也就是说，当程序已经开始了的时候），当 $\neg ok$ 成立的时候, P 就什么也不能说明了。

上面的 H1 条件是和同时满足下面两个法则等价的：

《skip;-unit》 $skip; P = P$

《；-L-Zero》 $Forever; D = Forever$

任何设计 $P \vdash Q$ 都满足上面的三个法则。

对于上面的 H2 条件，它有另外一个定义：

$$H2[P[False / ok'] \Rightarrow P[True / ok']]$$

由于没有可以满足不确定性的规范，当 ok’是假的的时候 P 就是真的，因此当 ok’是真的的时候它一定也是真的。

H2 健康条件是与满足下述法则等价的：

《；-H2-R-Unit》 $P; J = P$

《J-def》 $J \triangleq (ok \Rightarrow ok') \wedge S' = S$

几个公开的问题：

(1)是否存在更多的我们应当定义的健康条件？

在这些 H1~4 健康的程序中，是否有不合理的？这有可能以附带条件导致的推断的方式发生吗？

(2)我们是否可以想出可以辨认所有程序（像这些 *fac*）的有趣的子集的确定性的健康条件？

3.2 语义

一个建议的语义是如下的：

$$k = true \vdash res' = k \wedge S' \setminus res = S \setminus res$$

$$v = true \vdash res' = v \wedge S' \setminus res = S \setminus res$$

$$e1 \odot e2 = \exists r_0 \bullet ((e1[r_0 / res']; e2)) \sqcap ((e2[r_0 / res']; e1)); res := res \odot r_0$$

$$\mathrm{Re}t(e) \triangleq \exists r_0 \bullet e[r_0 / res']; res := r_0$$

$f(e) \triangleq \exists r_0 \bullet e\ r_0\ \ res\ \ \ p\ r_0\ \ v$, where $f(v) \triangleq p$

$$v := e = True \vdash \exists r_0 \bullet e[r_0 / res']; v' = r_0 \wedge S' \setminus v = S \setminus v$$

$$Skip = true \vdash S' = S$$

p;q = $\exists S_n, S_m \bullet P[S_n / S'] \wedge Q[S_m, S_n / S', S] \wedge R[S_m / S]$

$$\exists O_m \bullet p[O_m / O'] \wedge q[O_m / O]$$

$$p \lhd c \rhd q \;=\; \exists r_0 \bullet c[r_0 / res']; r_0 \wedge p \vee \neg r_0 \wedge q$$

$$c \circledast p \;=\; \mu W \bullet (P; W) \lhd c \rhd Skip$$

几个公开的问题：

(1) 我们有很多这种形式的定义：$\exists res_0 \bullet \ldots$ 我们是否可以找到一个简化所有这些的标示？

(2) 这些定义都是健康的吗？

4 基本法则

公开的问题：有哪些简单语言的程序法则仍然适用？哪些法则还需要改进？

下面我们尝试证明以下法则：[7][8]

（1）《skip-;-alt》

$skip_A = x :=_A x$

策略：从右至左进行证明

证明：

$x :=_A x$

= “:=-def , A={S’,S}”

$x' = x \wedge S' \backslash x' = S \backslash x$

= “merge”

S’=S

= “skip-def , A={S,S’}”

$skip_A$

（2）《skip-;-unit》

Skip;P = P

策略：从左至右进行证明

证明：

skip; P

= “skip-def”

S’ =S;P

= “; - def”

$\exists S_m \bullet S_m = S \wedge P[S_m / S]$

= “$\exists -1pt$”

$(P[S_m / S])[S / S_m]$

= “substitution ‘inverse’”

P

(3)《; -skip-unit》

P;skip = P

策略：从左至右进行证明

证明：

P;skip

= “skip-def”

P; S’=S

= “;-def”

$\exists S_m \bullet P[S_m / S'] \wedge S' = S_m$

= “$\wedge - comm$”

$\exists S_m \bullet S' = S_m \wedge P[S_m / S']$

= “$\equiv - comm$”

$\exists S_m \bullet S_m = S' \wedge P[S_m / S']$

= “$\exists - 1pt$”

$(P[S_m / S'])[S' / S_m]$

= “substitution ‘inverse’”

P

(4)《; -assoc》

$P;(Q;R) = (P;Q);R$

策略：从右边和左边同时推导，直到出现同样的命题

证明：

$P;(Q;R)$

= “; -def, $\{S, S'\} = out_\alpha Q \cup in_\alpha R$”

$P;(\exists S_m \bullet Q[S_m / S'] \wedge R[S_m / S])$

= “;-def, $\{S, S'\} = out_\alpha P \cup in_\alpha Q$”

$\exists S_n \bullet P[S_n / S'] \wedge (S_m \bullet Q[S_m / S'] \wedge R[S_m / S])[S_n / S]$

= “def. of substitution, $S \neq S_m$”

$\exists S_n \bullet P[S_n / S'] \wedge \exists S_m \bullet Q[S_m / S'][S_n / S] \wedge R[S_m / S][S_n / S]$

= “$\wedge - \exists - distr, S_m \notin P[S_n / S']$”

$\exists S_n, S_m \bullet P[S_n / S'] \wedge Q[S_m / S'][S_n / S] \wedge R[S_m / S][S_n / S]$

= “subst-seq, $S \notin S_m$”

$\exists S_n, S_m \bullet P[S_n / S'] \wedge Q[S_m, S_n / S', S] \wedge R[S_m / S][S_n / S]$

= “subst-comp, $S_m[S_n / S] = S_m$ ”

$\exists S_n, S_m \bullet P[S_n / S'] \wedge Q[S_m, S_n / S', S] \wedge R[S_m / S]$

$(P;Q);R$

= “;-def, $\{S, S'\} = out_\alpha Q \cup in_\alpha R$”

$(\exists S_n \bullet P[S_n / S'] \wedge Q[S_n / S]);R$

= “;-def, $\{S, S'\} = out_\alpha Q \cup in_\alpha R$”

$\exists S_m \bullet (\exists S_n \bullet P[S_n / S'] \wedge Q[S_n / S][S_m / S']) \wedge R[S_m / S]$

= “def. of substitution, $S' \neq S_n$”

$\exists S_m \bullet (\exists S_n \bullet P[S_n / S'][S_m / S'] \wedge Q[S_n / S][S_m / S']) \wedge R[S_m / S]$

= “$\wedge - \exists - distr, S_n \notin R[S_m / S]$”

$\exists S_m, S_n \bullet P[S_n / S'][S_m / S'] \wedge Q[S_n / S][S_m / S'] \wedge R[S_m / S]$

= “subst-seq, $S' \notin S_n$”

$\exists S_m, S_n \bullet P[S_n / S'][S_m / S'] \wedge Q[S_n, S_m / S, S'] \wedge R[S_m / S]$

= “subst-comp, $S_n[S_m / S'] = S_n$”

$\exists S_n, S_m \bullet P[S_n / S'] \wedge Q[S_m, S_n / S', S] \wedge R[S_m / S]$

= “除了些许的顺序不一致，右边 =左边”

（5）《$\lhd\rhd - false$》

$P \lhd False \rhd Q = Q$

证明：

$P \lhd False \rhd Q$

= “def. of $\lhd\rhd$”

$False \wedge P \vee \neg False \wedge Q$

= “$\wedge - comm,\ false - neg, \wedge - zero, \wedge - unit$”

$False \vee Q$

= “$\vee - comm, \vee - unit$”

Q

（6）《$\lhd\rhd - seq$》

$(P \lhd c \rhd Q); R = (P; R) \lhd c \rhd (Q; R)$

策略：同时从左边和右边进行推导，最终得到等价的命题。

证明:

$(P \lhd c \rhd Q); R$

= “$\lhd\rhd$ 的定义”

$(c \wedge P \vee \neg c \wedge Q); R$

= “;的定义”

$\exists S_m \bullet (c \wedge P \vee \neg c \wedge Q)[S_m / S'] \wedge R[S_m / S]$

= “命题的替换”

$\exists S_m \bullet (c \wedge P)[S_m / S'] \vee (\neg c \wedge Q)[S_m / S'] \wedge R[S_m / S]$

$(P;R) \lhd c \rhd (Q;R)$

= “两次使用；的定义，$\lhd\rhd$ 的定义”

$\exists T_m, W_m \bullet c \wedge P[T_m / T'] \wedge R[T_m / T] \vee \neg c \wedge Q[W_m / W'] \wedge R[W_m / W]$

= “代入特定值， $T_m = S_m, W_m = S_m$”

$\exists S_m \bullet (c \wedge P)[S_m / S'] \vee (\neg c \wedge Q)[S_m / S'] \wedge R[S_m / S]$

= “现在右边和左边已经被推导成了相同的命题”

（7）《$^{\lhd\rhd} - swap$》

$P \lhd c \rhd (Q \lhd d \rhd R)$ = $(P \lhd c \rhd Q) \lhd c \vee d \rhd R$

策略：同时从左边和右边进行推导，得到一个同样的命题。

证明：

$P \lhd c \rhd (Q \lhd d \rhd R)$

= “$\lhd\rhd$ 的定义”

$P \lhd c \rhd (d \wedge Q \vee \neg d \wedge R)$

= “$\lhd\rhd$ 的定义”

$c \wedge P \vee \neg c \wedge (d \wedge Q \vee \neg d \wedge R)$

= “$\wedge - \vee - distr, \wedge - comm$”

$c \wedge P \vee \neg c \wedge d \wedge Q \vee \neg c \wedge \neg d \wedge R$

$(P \lhd c \rhd Q) \lhd c \vee d \rhd R$

= “两次使用 $\lhd\rhd$ 的定义，德摩根定律”

$(c \vee d) \wedge (c \wedge P \vee \neg c \wedge Q) \vee (\neg c \wedge \neg d) \wedge R$

= “ $\wedge - \vee - distr$ ”

$c \wedge P \vee c \wedge d \wedge P \vee (\neg c \wedge Q \wedge d) \vee \neg c \wedge \neg d \wedge R$

= “ $\vee - \wedge - absorb, \wedge - comm$ ”

$c \wedge P \vee \neg c \wedge d \wedge Q \vee \neg c \wedge \neg d \wedge R$

= “左边 = 右边”

（8）《 $\lhd\rhd - subst$ 》

$P \lhd c \rhd Q[e/x]$ = $P[e/x] \lhd c[e/x] \rhd Q[e/x]$

策略：从左至右进行证明。

证明：

$(P \lhd c \rhd Q)[e/x]$

= “ $\lhd\rhd$ 的定义”

$(c \wedge P \vee \neg c \wedge Q)[e/x]$

= “命题的替换”

$(c \wedge P)[e/x] \vee (\neg c \wedge Q)[e/x]$

= “命题的替换”

$c[e/x] \wedge P[e/x] \vee (\neg c)[e/x] \wedge Q[e/x]$

= “命题的替换”

$c[e/x] \wedge P[e/x] \vee \neg c[e/x] \wedge Q[e/x]$

= “ $\lhd\rhd$ 的定义”

$P[e/x] \lhd c[e/x] \rhd Q[e/x]$

（9）《$:=-seq$》

$x:=e;x:=f$ = $x:=f[e/x]$

策略：从左至右进行证明。

证明：

$x:=e;x:=f$

= “$:=-def, A=\{S,S'\}$”

$x'=e\wedge S'\backslash x'=S\backslash x; x'=f\wedge S'\backslash x'=S\backslash x$

= “；-def, $A=\{S,S'\}$”

$\exists S_m \bullet (x'=e\wedge S'\backslash x'=S\backslash x)[S_m/S']\wedge(x'=f\wedge S'\backslash x'=S\backslash x)[S_m/S]$

= “注意到 e 没有带撇的变量，我们在这里进行替换”

$\exists S_m \bullet x_m=e\wedge S_m\backslash x_m=S\backslash x\wedge x'=f[S_m/S]\wedge S'\backslash x'=S_m\backslash x_m$

= “$\exists -1pt, S_m\notin e,S$”

$x'=f[e,S\backslash x/x,S\backslash x]\wedge S'\backslash x'=S\backslash x$

= “忽略掉$[S\backslash x/S\backslash x],:=-def$”

$x:=f[e/x]$

（10）《$:=-swap$》

$x:=e;y:=f$ = $y:=f[e/x];x:=e,y\notin e$

策略：从左至右进行证明。

证明：

$x:=e;y:=f$

= “$:=-def$”

$x'=e\wedge S'\backslash x'=S\backslash x; y'=f\wedge S'\backslash y'=S\backslash y$

= “；的定义，替换”

$\exists S_m \bullet x_m=e\wedge S_m\backslash x_m=S\backslash x\wedge y'=f[S_m/S]\wedge S'\backslash y'=S_m\backslash y_m$

= “$\exists -1pt$，注意到$y_m=y$”

$x'=e\wedge y'=f[e,S\backslash e/x,S\backslash x]\wedge S'\backslash_{x',y'}=S\backslash_{x,y}$

= “$:=-def$，忽略掉$[y,S\backslash_{x,y}/y,S\backslash_{x,y}]$”

$x,y:=e,f[e/x]$

(11)《sim-:=-def》

$\vec{x}:=_A \vec{e}\triangleq x_1'=e_1\wedge\ldots\wedge x_n'=e_n\wedge S'\backslash x_1'\ldots x_n'=S\backslash x_1\ldots x_n$

(12)《$\lhd\rhd -true$》

$P\lhd True\rhd Q$ = P

证明：

$P\lhd True\rhd Q$

= “$\lhd\rhd$的定义”

$True\wedge P\vee\neg True\wedge Q$

= “$\wedge-unit, false-neg, \wedge-zero$”

$P\vee False$

= “$\vee-unit$”

P

5 阶乘的例子

下面是以此语言定义一个阶乘函数的简单例子：

$$fac(n) \triangleq x := n; f := 1;$$

$$(x > 1) \circledast (f := x * f; x := x - 1);$$

$$\mathrm{Re}t(f)$$

我们看到了这个声明：

$$y := fac(3)$$

让 y 得到了值 6，但是也将 x 和 f 分别变化为了 1 和 6 。因为它带有附带影响，但 fac 是确定性的，由于只要它的参数表达是确定性的，它的非确定性影响也就大都是一样的：

$$v := fac(e)$$

$$= \quad v' = e! \wedge f' = e! \wedge (x' = 1 \lhd e \geq 1 \rhd x' = e) \wedge S'\backslash_{v,f,x} = S\backslash_{v,f,x}$$

（这里我们假设 e 是一个常量或者单一变量，假设它自己没有附带影响。）

5.1 对数阶乘

下面是一个众所周知的代数法则，无可置喙：

$$x + x = 2 * x$$

但是现在让我们来考虑一个叫作 $logfac$ 的 “对数”版本的阶乘函数，它计数了加运算的次数：

$$\log fac(n) \triangleq x := n; f := 1;$$

$$(x > 1) \circledast (f := x * f; x := x - 1);$$

$$count := count + 1;$$

$$\mathrm{Re}t(f)$$

现在我们来问一个问题：上述两个程序状态是一样的吗？

$$y := \log fac(z) + \log fac(z)$$

$$y := 2 * \log fac(z)$$

我们可以发现代数规则在这里不适用了——*logfac(z)*并不是一个值依赖于 z 的数,而是一个改变了 x ,f 和 *count* 的值的程序状态！第一个状态会给 *count* 加 2,第二个只会给 *count* 加 1。但它们都会将 y 变为 $z!$。

这个例子就说明了附带影响的影响以及它们如何阻止我们使用原本非常棒的代数规则，即使看起来这样的规则似乎是能用的。

5.2 依赖于历史行为的附带影响

现在让我们来考虑两个计算不同数值结果、但都通过不同的方法修正了 “hash” 的数值的函数。

$$f(x) \triangleq hash := hash + 1; \mathrm{Re}\,t(x+1)$$

$$g(x) \triangleq hash := 2 * hash; \mathrm{Re}\,t(2 * x)$$

另外一个众所周知的代数规则是这样陈述的：

$$a + b = b + a$$

那么对于以下两个程序我们该怎么评判呢？

$$y := f(1) + g(2)$$

$$y := g(2) + f(1)$$

它们都让 y 拥有了值 6，但是 *hash* 的值又发生了什么变化呢？

先对 f(1)进行操作，然后是 g(2)，此时 $hash' = 2 * (hash + 1)$；

先对 g(2)进行操作，然后是 f(1)，此时 $hash' = (2 * hash) + 1$。

因此，这很简单，在这里代数规则不适用，就像之前的例子一样，此外，两个程序给出了不同但可预测的结果。

不！这是错误的！

这两个程序看起来是不同的,但是一个优化了的 C 语言的编译器并没有义务按照子表达式被写下来的顺序来执行。顺序也许会被保持，或者它可能被保留，又或许一些别的准则将它们按照一些典范的顺序来分类。

我们并不能分辨——上述这两个程序也许有着完全相同的行为表现或者总是不同，但根据优化的改变程度或者正在使用的是那哪个版本的编译器这一点也会变化。这就是为什么如果我们想要有能力为一种语言赋予一个符合真实的语义，语义中的不确定性是有很大影响的。

6 语义的健康条件检验

检验语言语义是否符合健康条件：
如果是，给出证明；
如果不是，对于语义或者健康条件给出修正建议。

我们想要证明所有的定义都是健康的，也就是说，$H1(H2(P)) = P$，至少，我们要记住这种形式的定义 $P \vdash Q$ 在 P 和 Q 中都没有提到 ok 或者 ok’ 的时候，是健康的！

对于下面一个部分中的每个定义，我们都需要检验它的健康性。

6.1 k 是否健康?

$$k = True \vdash res' = k \wedge S' \backslash res = S \backslash res$$

显然，它是健康的，因为它的形式符合一个设计的样式。

6.2 v 是否健康？

$$v = True \vdash res' = v \wedge S' \backslash res = S \backslash res$$

是的，它是健康的，因为他的形式是符合一个设计的样式的。

6.3 $e1 \odot e2$ 是否健康？

$$e1 \odot e2 = \exists r_0 \bullet ((e1[r_0 / res']; e2) \mid\sim\mid ((e2[r_0 / res']; e1)); res := res \odot r_0$$

这个语义并不具有一个设计的样式，因此我们将作进一步的证明，如下：

$$H1(H2(e1 \odot e2)) = e1 \odot e2$$

但我们在这里假设 e1 和 e2 都是符合设计的样式。
证明：

引理 1： 如果 p 和 q 都是健康的，那么 $p \vee q$ 就是健康的。

证明：
H1-性质检验：

$$H1(p \vee q)$$

= *“ H1 的定义”*

$ok \Rightarrow (p \vee q)$

$= "def.of \Rightarrow"$

$\neg ok \vee (p \vee q)$

$= "\vee - \vee - distr"$

$(\neg ok \vee p) \vee (\neg ok \vee q)$

= *“因为 p 和 q 都是健康的”*

$p \vee q$

因此它满足 H1 健康条件！

H2-性质检验：

我们有两种不同的方法来做 H2 性质检验，我们会将它们都展示出来。

第一种方法：

$p \vee q; J$

= *“J 的定义”*

$p \vee q; (ok \Rightarrow ok') \vee S' = S$

= *“;的定义”*

$(p \vee q)[False / ok'] \exists ok_m, S_m \bullet (p \vee q)[ok_m, S_m / ok', S'] \vee (ok_m \Rightarrow ok') \vee S' = S_m$

= *“赋值，* $ok_m = False, S_m = S'$ *”*

$(p \vee q)[False / ok']$

= *“如果上式值为真的话 H2-性质就得到了证明。”*

我们在这里稍作停留，先开始展示第二种证明方法：

第二种方法：

$H2(p \vee q)$

= *“H2 的定义”*

$c \circledast p = \mu W \bullet (P; W) \lhd c \rhd Skip \ (p \vee q)[False / ok'] \Rightarrow (p \vee q)[True / ok']$

= *“接下来就很容易知道，如果⇒ 前面的部分的值为假，那么* $H2(p \vee q)$ *就是真的，并且它满足 H2 健康条件！”*

从上述两种证明方法中我们知道了，不论 $(p \vee q)[False / ok']$ *的赋值是假或是真，*

$(p \vee q)$ *都将是 H2 健康的！*

因此，不论怎样，它都符合 H2 健康条件！引理 1 的证明至此完成！

因为这一部分 $res := res \odot r_0$ 根据其定义符合一个设计的样式，因此它是健康的，我们可以暂时不去管这一部分了。

让我们现在来关注一下；之前的部分,

$$\exists r_0 \bullet ((e_1[r_0 / res']; e_2) \vee (e_2[r_0 / res']; e_1))$$

= “$\exists - distr$”

$$(\exists r_0 \bullet e_1[r_0 / res']; e_2) \vee (\exists r_0 \bullet (e_2[r_0 / res']; e_1))$$

= “首先让我们来关注∨之前的部分,如果 r_0 不存在，要使得 $e_1[r_0 / res']; e_2$ 是健康的，那么 $e_1[r_0 / res']; e_2$ 无论怎样都不可能是健康的，但 $e_1[r_0 / res']; e_2$ 是健康的，这是因为 e_1 和 e_2 都符合设计的样式，而且 ok 或者 ok’ 在 $e_1[r_0 / res']$ 都未出现，那么用 r_0 代替 res’ 是不会改变原命题的健康性的！ 因此第一部分是健康的！ 并且通过同样的方法我们知道了第二部分也是健康的！ 通过引理 1，我们知道上面的整个语义都是健康的!”

下面看一下 后面的第八小部分 $p;q$ 的证明，我们就得到了 $e_1 \odot e_2$ 的健康性。

到这里，证明工作就已经完成了！

6.4 $\mathrm{Re}t(e)$ 是否健康？

$$\mathrm{Re}t(e) = \exists r_0 \bullet e[r_0 / res']; res := r_0$$

我们将这个证明通过三个步骤展示出来：

首先，如果 e 是健康的，那么 $e[r_0 / res']$ 就是健康的，这是因为 H1 和 H2 都是只与 ok 和 ok’ 有关的，那么用 res’ 来代替 r_0 不会改变它的健康性。

第二步，我们将证明 $res := r_0$ 是健康的：

因为这一部分和 v := e 有着一样的结构（设计的样式），因此它是健康的。

第三步，我们将证明，如果 P 和 Q 都是健康的，那么 P;Q 就是健康的：

在后面的 p;q 部分中我们将会证明这一点。

第四步，证明如果 P 是健康的，那么 $\exists r_0 \bullet P$ 就是健康的:(我们把这个标注为引理 3)。

因为P是健康的，那么我们就可以将它写成 $Q \vdash R$ 的形式。

$\exists r_0 \bullet Q \vdash R$

= “ ‘设计’的定义”

$\exists r_0 \bullet ok \wedge Q \Rightarrow ok' \wedge R$

= “$\Rightarrow$的定义，德摩根定律”

$\exists r_0 \bullet \neg ok \vee \neg Q \vee ok' \wedge R$

= “$\exists - \vee - distr$”

$(\exists r_0 \bullet \neg ok) \vee (\exists r_0 \bullet \neg Q) \vee (\exists r_0 \bullet ok' \wedge R)$

= “ok，ok’都不是 r_0, 因此它们可以被移动到量词外面”

$\neg ok \vee (\exists r_0 \bullet \neg Q) \vee ok' \wedge (\exists r_0 \bullet R)$

= “德摩根定律”

$\neg(ok \wedge \neg(\exists r_0 \bullet \neg Q)) \vee ok' \wedge (\exists r_0 \bullet R)$

= “$\vdash$的定义”

$\neg(\exists r_0 \bullet \neg Q) \vdash (\exists r_0 \bullet R)$

由于它是 $S \vdash T$ 的形式，符合设计的样式，所以它是健康的！

根据上面的四个步骤，我们知道如果e是健康的，那么Ret(e) 就是健康的。

6.5 $f(e)$是否健康？

$f(e) = \exists r_0 \bullet e[r_0 / res]; p[r_0 / v]$

其中， $f(v) \triangleq p$

基于 Ret(e) 的证明，我们将这个证明分成三个步骤来实现：

首先，我们证明 $\exists r_0 \bullet e[r_0 / res']$ 是健康的：

$\exists r_0 \bullet e[r_0 / res']$

= “因为 $e[r_0 / res']$ 是健康的，用反证法来证明，如果对于任意变量 r_0，$e[r_0 / res']$ 都是不健康的，那么 $e[r_0 / res']$ 是健康的这一点就不能成立，因此 $\exists r_0 \bullet e[r_0 / res']$ 是健康的。”

第二步，证明如果 p 和q都是健康的，那么 p;q 就是健康的。这一步在此处省略，请参看后面的第八部分.

第三步，证明 $p[r_0 / v]$ 是健康的:

假设 p 是健康的.因为 H1 和 H2 都只是与 ok 和 ok’ 有关的，那么用 res’ 来代替 r_0 并不会改变它的健康性质。

那么至此，证明就已经完成了，我们也知道了如果 p 是健康的，那么 f(e) 就是健康的!

6.6 $v := e$ 是否健康？

$$v := e = True \vdash \exists r_0 \bullet e[r_0 / res']; v' = r_0 \wedge S' \setminus v' = S \setminus v$$

因为它满足一个设计的样式，所以它是健康的。

6.7 $Skip$ 是否健康？

$$Skip = true \vdash S' = S$$

因为它满足一个设计的样式，所以它是健康的。

6.8 $p;q$ 是否健康？

$$p;q = \exists O_m \bullet p[O_m / O'] \wedge q[O_m / O]$$

H1 性质检验：

$ok \Rightarrow (p;q)$

= “$\Rightarrow$的定义”

$\neg ok \vee (p;q)$

= “；的定义”

$\neg ok \vee (\exists O_m \bullet p[O_m / O'] \wedge q[O_m / O])$

= “由于 O_m 在 ok 中并不出现，我们在这里就可以将 ok 移入”

$\exists O_m \bullet \neg ok \vee p[O_m / O'] \wedge q[O_m / O]$

= “$\vee - \wedge - distr$”

$\exists O_m \bullet (\neg ok \vee p[O_m / O']) \wedge (\neg ok \vee q[O_m / O])$

= “$\Rightarrow$的定义”

$\exists O_m \bullet (ok \Rightarrow p[O_m / O']) \wedge (ok \Rightarrow q[O_m / O])$

= “假设 p 和 q 都是健康的，因为 ok, ok’ 或 ok_m 都不出现在 p 或 q 中，所以 $p[O_m / O]$ 和 $q[O_m / O]$ 都是健康的”

$\exists O_m \bullet p[O_m / O'] \wedge q[O_m / O]$

= “；的定义”

$p;q$

因此，如果 p 和 q 都是健康的，那么它就是 H1-健康的。

H2-性质检验:

$(p;q);J$

= “;- assoc”

$p;(q;J)$

= “假设 q 是健康的”

$p;q$

因此如果 q 是健康的，它就是 H2 健康的。

引理 2：如果 p 和 q 都是健康的，那么 $p \wedge q$ 就是健康的。

证明：

H1 性质检验：

$ok \Rightarrow p \wedge q$

= *“$\Rightarrow$ 的定义”*

$\neg ok \vee p \wedge q$

= “$\vee - \wedge - distr$ ”

$(\neg ok \vee p) \wedge (\neg ok \vee q)$

= “$\Rightarrow$ 的定义”

$(ok \Rightarrow p) \wedge (ok \Rightarrow q)$

= “因为 p 和 q 都是健康的”

$p \wedge q$

H2 性质检验：
我们有两种不同的方法来进行这个证明，我们会将它们都陈述出来：
第一种方法：

$p \wedge q; J$

= “J 的定义”

$p \wedge q; (ok \Rightarrow ok') \wedge S' = S$

= “; 的定义”

$\exists ok_m, S_m \bullet (p \wedge q)[ok_m, S_m / ok', S'] \wedge (ok_m \Rightarrow ok') \wedge S' = S_m$

= “特定值代入， $ok_m = False, S_m = S'$ ”

$(p \wedge q)[False / ok']$

= “如果上式赋值为真，那么证明就完成了。”
我们在这里暂停，先看看第二种方法：
第二种方法：

$H2(p \wedge q)$

= “H2 的定义”

$(p \wedge q)[False / ok'] \Rightarrow (p \wedge q)[True / ok']$

= “然后就可以很容易知道，如果$\Rightarrow$之前的部分的值为假， 那么 $H2(p \wedge q)$ 就是真的，命题就是 H2-健康的!”

从上述两种方法中我们知道不论 $(p \wedge q)[False / ok']$ 的值是真还是假，$(p \wedge q)$ 都将是 H2-健康的!
那么原命题就是 H2 健康的！引理 2 的证明至此完成！

6.9 $p \lhd c \rhd q$ 是否健康？

$$p \lhd c \rhd q \quad = \quad \exists r_0 \bullet c[r_0 / res'];r_0 \wedge p \neg r_0 \wedge q$$

由于 H1 和 H2 都只是与 ok 和 ok’有关的，因此 r_0 被 res' 所代替并不会改变它的健康性质。

$$\exists r_0 \bullet e[r_0 / res']$$

= “因为 $e[r_0 / res']$ 是健康的，我们用反证法来证明，如果对于任何变量 r_0 来说，$e[r_0 / res']$ 都是不健康的，那么 $e[r_0 / res']$ 是健康的这一点就不能成立了，因此 $\exists r_0 \bullet e[r_0 / res']$ 是健康的。”

因此我们知道了如果 c 是健康的，；前面的部分就是健康的。现在让我们看看第二部分的健康性（在；之后的部分）：

如果 r_0 是真的那么第二部分就等价于 p，否则它就等价于 q，而且我们知道 p 和 q 都是健康的，因此第二部分是健康的！

并且我们知道如果 p 和 q 都是健康的那么 p;q 也就是健康的。

那么 $p \lhd c \rhd q$ 的健康性就在 c、q 和 p 的健康性都成立的条件下成立！证明至此完成！

6.10 $c \circledast p$ 是否健康？

$$c \circledast p = \mu W \bullet (P;W) \lhd c \rhd Skip$$

这里我们假设这一条语义是健康的。（事实上它也确实是，但因为证明过程太过复杂，我们在这里不做过多说明。）

结论

在这片论文中，我按照预期完成了主要目标：

首先，攻克了开始阶段提到的两个难点：定义了所有的健康条件，并且抓住了程序子集合不确定性特点；另外，也把握住了创新点，即将一个 C 语言（不确定性表达）语义中的复杂部分形式化，而不是坚持使用开始阶段的语言的简单子集合。

具体来讲，我在文章中首先给出了一个建议使用的样本语言；然后较详细地给出健康性及健康条件的定义；接下来，从语法、细化、健康性和语义等四个方面探索出一套具体适用于已提到的样本语言的语义；第四步，给出了十一条基本一阶谓词演算法则，并适用 $U(TP)^2$ 进行辅助证明；在第五章中，给出了具体例子，通过阶乘、对数阶乘以及依赖于历史行为的附带影响这三个层面的粒子来展示和说明语义不规范的多种影响和它带来的麻烦，侧面反映本课题的重要意义！最后，针对我们给出的十条新的语义逐条进行了健康性的检验，并在检验过程中给出了语义的完善方案。

展望

本课题还有很大的完善和发展空间，未来还有很长的路要走，未来的工作计划罗列如下：

（1） 可以尝试解决本文某些地方提到的公开的问题；

（2） 可以研究对于类似 C 的语言中简单语言规律（App.A）中还有哪些也是真的：如果有，请给出证明；如果没有，说明哪些规律的修改版本（使之成为真的）；

（3） 探究我们是否能够添加额外的可以分别出有着有趣性质的程序或者表达式的健康条件：例如那些确定性的或者没有附带影响的条件；

（4） 当我们已经有了一个非确定性的程序时，证明它可能会怎样被变换到一个确定性的（安全的）程序，并且该变换被证明为正确的可能性的存在性；

（5） 探究怎样在两种语言（简单语言和 C 类似语言）的语义之间建立一个 Galois 链接。

致谢

感谢，是一个人在获得他人的帮助、接受他人给予的鼓励或他人提供的方便、恩惠、利益，使自己得到提高、进步、完善、圆满、成功之后，出于内心的感激之情，用言行向对方表达谢意的行为。

自己一直都不喜欢“谢谢”这个词，因为总觉得这个词是没有意义的，但我还是很感谢北航的毕设论文要求，因为它里面有致谢这一项，让我将你们的恩情镌刻、铭记！

感谢我的导师 Andrew Butterfield 先生！一直都记得，在我迷茫不知所已的时候，您耐心地帮我指出选择的方向；一直都记得，在我心中稍感遗憾觉得无缘选择形式化方法这门课程的时候，是您主动提出将我加到 blackboard 上，让我能够同步线下学习这门后来证明是我一直要寻找的“逻辑”课；后来我有幸成为这门课程的一名学生真是上学期最正确的选择之一了；一直都记得您耐心的答疑；还有您答应带一个学籍已经不在圣三一的学生的毕业设计时的无私奉献和以学生为本！感谢您认真地为我选定题目并在之后一步一步带领我实现这个课题，您的一次次修改，一次次非常详细具体的新任务的布置，一次次如斯认真的评注都激励着我不断拼搏、追求自己喜欢的知识和技能！我倍感荣幸，我也为自己在过程中有些时候不够努力而惭愧，我希望未来我会越来越好，更加主动和富有创造力，在这个项目的进一步工作中不辜负您的付出！

感谢我的导师漆毅老师！您是我们的数学分析任课老师，一直以来您给大家的印象都是一个非常认真负责、非常和蔼可亲、为学生着想的老师，记得我当年是数学分析课代表，这个课代表给我留下了好多好多美好的回忆！感谢您对我们的谆谆教诲和对我这个课代表的信任！感谢您在我学习和选择过程中迷茫的时候百忙之中还抽出时间为我分析形势、提出宝贵的建议！感谢您每一次考期答疑过程中如斯兢兢业业，为我解答问题的稿纸写了一页又一页！感谢您在我大学四年中对我大大小小的一次又一次的帮助和关照！感谢您为我的毕业设计相关事宜的付出和帮助！我希望我以后会在学术的道路上走得更加坚定，不辜负您的期望！

当然！我还要感谢我的姥姥和姥爷！是你们在我灰心气馁和不高兴的时候帮我调整情绪、排忧解难，还为我打气，每次和你们通完电话我都觉得生活特别美好，

觉得自己有了奋斗的方向！请你们放心，我一定会好好努力，追寻着自己喜爱的方向，在追求真善美的道路上坚定不移地走下去，做最好的自己！不辜负你们的期望！

我还要感谢“2014 南开数理逻辑研讨会”上的各位前辈老师们，谢谢你们对我的未来方向和毕设方向提出了宝贵的意见！感谢中国科学院的冯琦老师一直以来对我的鼓励和帮助！感谢上海交通大学的傅育熙老师百忙之中抽出时间对我的毕业设计提出了建设性的意见！

之前使用的是 latex 版本，过程中有很多不熟练的地方，即使最终放弃 latex 版本，还是要特别感谢之前的过程中同学们、老师们对我的帮助！

还要感谢《三国》剧组，是你们让我在这个毕业季还拥有一段难忘的美好的回忆！

我想，毕业设计应该是一个人大学四年学术生涯的一个缩影和部分总结，感谢到这里，已经停不下来了……我决定在这里感谢更多的人：

感谢武汉大学的曾宪武教授，感谢您在我大一暑假的时候不顾炎夏酷暑，为我传授您毕生所学之经验！感谢您和家人为当时独在异乡的我如亲人般的照顾！

感谢北京大学的王敏中教授，感谢您在我刚开始学习数学分析的时候与我分享您当年的学习感受和经历！

感谢教零，感谢沙河，感谢北航，感谢课程设置，感谢所有我的任课老师，感谢所有对我的学术有所启发的人和事！

感谢我的班主任郭炳晖老师！感谢您之前对我的鼓励，感谢您在美赛建模时候对我的指导和帮助！感谢您为了我的发展做出的付出！

感谢我的导师郭雷院士！感谢您在大三时候让我们一起去听您的讨论班，增长见识，扩宽视野！感谢您几多次在我迷茫不知何去何从时耐心的教导！感谢您在听闻我的学习兴趣并不主要是系统控制时候给予的理解！感谢您当时为了我的兴趣方向和我的更好的发展为我提供您的相关领域的同事的联络！感谢命运，我很庆幸有缘可以选择您作为我的本科阶段的导师！

感谢都柏林圣三一学院的 Pete 教授！一直都记得，您始终面带微笑，解答每一个学生的每一个问题！因为我是交换生，当时有很多的细小的问题和麻烦，您总是那么 ready to help，总是急学生之所急，想学生之所想，我想 trinity 就是因为有很多像您这样的行政人员才会让学生们都有一个好的发展吧！

感谢都柏林圣三一学院的 Stalker 先生，感谢您在我对于未来选择迷茫不知所措的时候热心地帮我，为我提供可以去询问的朋友的联络！

感谢都柏林圣三一学院计算机学院符号编程课程的老师（实在抱歉我想不起他的名字了），感谢您在百忙之中抽空听我诉说疑难，为我提供您宝贵的意见！

感谢怀校长，感谢老师们，感谢辅导员们，感谢华罗庚班，感谢北航！……

突然觉得大学四年中要感谢的人和事是在太多了，若是再继续写下去就是喧宾夺主了啊！好的，感谢命运，感谢上苍！……

致谢暂停于此……

参考文献

【1】http://see.xidian.edu.cn/cpp/html/4.html

【2】http://baike.baidu.com/view/1219.htm

【3】[Mis04] MISRA Consortium. *MISRA-C:2004—Guidelines for the use of the C language in critical systems*, 2004.

【4】http://en.wikipedia.org/wiki/Unifying_Theories_of_Programming

【5】[CC+00] C and c++ coding standards, bssc(2000)1 issue, 2000.

【6】[HH98] C.A.R. Hoare and Jifeng He. *Unifying Theories of Programming*. Prentics-Hall, 1998

【7】CS4003-Reference-Main, Andrew Butterfield, Oct. 16th, 2013.

【8】Lecture slides of CS4003, Andrew Butterfield.