改进的流不敏感的类型限定词推断

李慧松[1,2] 许智武[1] 陈海明[1]
(中国科学院软件研究所计算机科学国家重点实验室 北京 100190)[1]
(中国科学院大学 北京 100049)[2]

摘 要 类型限定词可以精化标准类型,提高类型系统的表达能力。流不敏感的类型限定词推断已被用于CQual架构,以提高C程序的质量。然而,类型转化会影响类型限定词推断的有效性。首先,展示了一种允许类型转化的程序语言和流不敏感的限定词推断系统;其次,提出了变量参与的限定词推断系统,引入了联合类型并给出约束求解算法;最后,证明了推断的正确性并展示了一些实例运行结果。

关键词 类型转化,类型推断,限定词,流不敏感,联合类型
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.034

Improved Flow-insensitive Type Qualifier Inference

LI Hui-song[1,2] XU Zhi-wu[1] CHEN Hai-ming[1]
(State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]
(University of Chinese Academy of Sciences, Beijing 100049, China)[2]

Abstract Type qualifiers can refine the standard types and improve the expressivity of type systems. Flow-insensitive type qualifier inference has been used in the CQual framework to improve the quality of C programs. Type casts, however, will affect the effectiveness of type qualifier inference. First a language allowing type casts and its flow-insensitive qualifier inference system were presented. Then this paper proposed a variable-involved inference system, introduced union types and given constraints solving algorithm. Finally, the soundness was proved and some case studies were pre-sented.

Keywords Type casts, Type inference, Qualifiers, Flow-insensitive, Union types

1 引言

类型限定词精化标准类型并提高类型系统的表达能力。通常,限定词被分为两类:值限定词(如 tainted 和 untainted)和引用限定词(如 const 和 nonconst)。值限定词针对表达式的值,而引用限定词则针对表达式的地址[6]。

特定地,限定词 tainted 和 untainted 被用来检查 C 程序的格式化字符串缺陷[17]。通常,可以被不受信任的用户(如网络用户)控制的数据应该被声明为 tainted。反之,格式化声明符(如打印函数 printf 的第一个参数)声明了后面参数的个数,故需为受信任(untainted)的数据。把 untainted 数据解释成 tainted 数据是安全的,反之则不然。因此,我们定义关系 untainted≤tainted。限定词 const 和 nonconst 被用于 C 程序的 const 推断[11]。通常,nonconst 内存单元可以被重新赋值,而 const 内存单元不能被重新赋值。故,把一个 nonconst 内存单元解释成 const 内存单元是安全的,反之则不然。因此,我们定义 nonconst≤const。

文献[11]中,Foster 提出了利用类型限定词来提高程序质量的架构。在该架构中,程序设计者可以自由声明限定词,并在程序中添加部分限定词注释。流敏感的限定词推断和流不敏感的限定词推断被用来推断其余的限定词和检查限定词注释的一致性。在流不敏感的限定词推断系统中,变量的限定类型在整个推断过程中保持一致,而在流敏感的限定词推断系统中,其会随着赋值等语句而改变。基于此架构,C 程序的静态分析工具 CQual 被开发且已有很多实际的应用[19,15,12,17,1,3]。

例 1 示例 C 程序片段

```
const int * x;
int * y;
int a,b;
a = (int) x; /* (1) */
b=a; /* (2) */
y=(int * ) b; /* (3) */
 * y= 5; /* (4) */
```

然而,类型转化会影响类型限定词推断的有效性。对于流不敏感的限定词推断,考虑例 1 里的程序片段。该程序是可以被 CQual 接受的。在此程序中,x 是一个常量指针。根据第(1)—(3)行,我们可以得到 x 和 y 拥有同样的值。因此,y 也应该为一个常量指针,且第(4)行的赋值应该被禁止。但

到稿日期:2013-11-05 返修日期:2014-02-16 本文受国家科技重大专项(2012ZX01039-004)资助。
李慧松(1989—),女,硕士生,CCF 学生会员,主要研究方向为程序的静态分析,E-mail:lihs@ios.ac.cn;**许智武**(1983—),男,博士,主要研究方向为程序语言、类型系统;**陈海明**(1966—),男,博士,研究员,CCF 高级会员,主要研究方向为形式语言与自动机理论、计算模型、形式规约和程序分析。

是 x 是一个常量指针的类型信息并不能通过变量 a 和 b 的类型 int 传递给 y 的类型。主要原因是，类型 int 没有 int * 承载的限定词信息多，导致在限定词推断过程中丢失了一些有用的限定词信息。注意第(1)行，x 的值在类型转化之后被赋值给 a，则 a 的值的实际类型是 int *。在第(2)行，a 的值被赋值给 b，那么 b 的值的实际类型也是 int *。因此，对于 a 和 b，在限定词的推断过程中，可以使用 int * 来代替 int。基于此，我们便可推断出 y 是一个常量指针。故，对于限定词的推断而言，int * 比 int 更适合作为 a 和 b 的类型，即，表达式最适合进行限定词推断的类型不一定是程序中声明的类型。

本文第 2 节介绍基本概念以及一个简单的源程序语言；第 3 节展示该语言的标准的流不敏感的限定词推断系统；第 4 节引入变量参与的限定词推断系统及联合类型，并基于此为每个表达式求解一个适合进行限定词推断的类型以及给出正确性的证明；最后展示了一些实例的运行结果。

2 基本概念

本文通过一个简单的源语言来展示基本的定义及概念。在引入源语言的基本表达式之前，我们先给出类型和限定类型的定义。类型 t 的生成文法是：

$t ::= \text{int} \mid \text{ref}(t) \mid t \to t$

其中，ref 是指针类型的构造子，→是函数类型的构造子。给定限定词集合 Q，限定类型 τ 的生成文法是：

$\tau :: q_c\sigma, q_c \in Q$

$\sigma :: = \text{int} \mid \text{ref}(\tau) \mid \tau \to \tau$

定义 1 含有类型转化和限定词注释的源程序语言：

$e ::= v$	值
$\mid e_1\, e_2$	应用
$\mid \text{let } x = e_1 \text{ in } e_2$	名字绑定
$\mid \text{ref } e$	引用
$\mid * e$	解引用
$\mid (t)\, e$	类型转化
$\mid e_1 := e_2$	赋值
$\mid \text{annot}(e, q_c)$	限定词注释
$v :: \text{n}$	整数
$\mid x$	变量
$\mid \lambda x{:}t.\, e$	函数

定义 1 定义了源语言的表达式，其中，$(t)e$ 将表达式 e 的类型转化为类型 t，annot(e, q_c) 声明 q_c 是表达式 e 的限定类型的最外层限定词。通常，限定词集合 Q 和限定词之间的关系≤形成部分序集合。为阐述方便，对于本文下面的所有例子，我们规定$(Q, \leqslant) = \{\text{untainted} \leqslant \text{tainted}\}$。

定义 2 限定子类型关系：

$$\frac{q_1 \leqslant q_2}{q_1\ \text{int} \leqslant q_2\ \text{int}} \quad (\text{Int}_{\leqslant})$$

$$\frac{q_1 \leqslant q_2\ \ \tau_1 = \tau_2}{q_1\ \text{ref}(\tau_1) \leqslant q_2\ \text{ref}(\tau_2)} \quad (\text{Ref}_{\leqslant})$$

$$\frac{q_1 \leqslant q_2\ \ \tau_3 \leqslant \tau_1\ \ \tau_2 \leqslant \tau_4}{q_1(\tau_1 \to \tau_2) \leqslant q_2(\tau_3 \to \tau_4)} \quad (\text{Fun}_{\leqslant})$$

定义 3 函数 $embed_q$ 和 $strip_q$ 的定义为：

$embed_q(\text{int}) = q\ \text{int}$ $\quad q$ fresh

$embed_q(\text{ref}(t)) = q\ \text{ref}(embed_q(t))$ $\quad q$ fresh

$embed_q(t_1 \to t_2) = q\ (embed_q(t_1) \to embed_q(t_2))$ $\quad q$ fresh

$strip_q(q\ \text{int}) = \text{int}$

$strip_q(q\ \text{ref}(\tau)) = \text{ref}(strip_q(\tau))$

$strip_q(q\ (\tau_1 \to \tau_2)) = strip_q(\tau_1) \to strip_q(\tau_2)$

例 2 是一个简单的源语言程序，其包含了指针类型 ref (int)和整型 int 之间的类型转化。该程序声明 x 指向 tainted 对象，z 指向 untainted 对象，并通过类型转化语句、赋值语句等将 x 的值赋给 z，意为把一个 tainted 对象解释成 untainted 对象，因此该程序是不安全的。

例 2 简单的源语言程序

let x = ref annot(0, tainted) in

let y = (int) x in

let z = (ref(int)) y in

annot($*z$, untainted)

类型限定词的部分序关系诱导定义 2 中限定类型上的限定子类型关系≤(为了表示方便，本文中≤同时表示限定词及限定类型上的关系)。$\tau_1 \leqslant \tau_2$ 表示把 τ_1 类型的对象解释成 τ_2 类型的对象是完全安全的。为了确保同一个引用(ref)单元的所有化名包含同样的限定词，规则($\text{Ref}_{\leqslant}$)要求 $\tau_1 = \tau_2$ ($\tau_1 \leqslant \tau_2$ 且 $\tau_2 \leqslant \tau_1$)，而不只是 $\tau_1 \leqslant \tau_2$。否则，我们便可把 ref(untainted int)类型的指针赋值给 ref(tainted int)类型的指针，并通过 ref(tainted int)类型的指针把 tainted 数据写到 untainted 的位置。规则($\text{Fun}_{\leqslant}$)表示函数类型的子类型关系与它的定义域是逆变的，与它的值域是共变的。在定义 3 中，函数 $embed_q$ 把标准类型映射到相应的带有新生限定词变量的限定类型，$strip_q$ 则把限定类型映射到它们的标准类型。限定词变量，记作 q，表示未知的限定词。

性质 1 $\forall\ \tau_1, \tau_2, \tau_3$，如果 $\tau_1 \leqslant \tau_2$ 并且 $\tau_2 \leqslant \tau_3$，那么 $\tau_1 \leqslant \tau_3$。

定义 4 可转化类型关系：

$\text{int} \triangleright \text{int}$	$(II_{\triangleright})$	$\text{int} \triangleright \text{ref}(t)$	$(IR_{\triangleright})$
$\text{ref}(t) \triangleright \text{int}$	$(RI_{\triangleright})$	$\text{ref}(t_1) \triangleright \text{ref}(t_2)$	$(RR_{\triangleright})$

定义 4 展示了源语言允许的类型转化。如果类型 t_1 被允许转化为类型 t_2，我们称 t_1 是 t_2 的可转化类型，记作 $t_1 \triangleright t_2$，即，类型 t_1 的值可以被解释为类型 t_2 的值。由于整数可以被解释成地址，反之亦然，因此，($\text{IR}_{\triangleright}$)和($\text{RI}_{\triangleright}$)表示指针类型和整数类型之间可以进行类型转化。由于任何地址都可以被解释成其它类型的地址，因此($\text{RR}_{\triangleright}$)表示任意类型的指针类型都可以相互转化。为阐述方便，我们禁止了其它类型转化，如函数类型之间的类型转化。

基于可转化类型关系，我们定义限定的可转化类型关系。首先定义 5 给出了关系 $\triangleright^q$ 的定义。

定义 5 关系 $\triangleright_q$ 的定义为：

$$\frac{q_1 \leqslant q_2}{q_1\ \text{int} \triangleright^q q_2\ \text{int}} \quad (II_{\triangleright^q})$$

$$\frac{q_1 \leqslant q_2}{q_1\ \text{int} \triangleright^q q_2\ \text{ref}(\tau)} \quad (IR_{\triangleright^q})$$

$$\frac{q_1 \leqslant q_2}{q_1\ \text{ref}(\tau) \triangleright^q q_2\ \text{int}} \quad (RI_{\triangleright^q})$$

$$\frac{q_1 \leqslant q_2}{q_1\ \text{int} \triangleright^q q_2(\tau_1 \to \tau_2)} \quad (IF_{\triangleright^q})$$

$$\frac{q_1 \leqslant q_2}{q_1(\tau_1 \to \tau_2) \triangleright^q q_2\ \text{int}} \quad (FI_{\triangleright^q})$$

$$\frac{q_1 \leqslant q_2}{q_1\ \text{ref}(\tau_1) \triangleright^q q_2(\tau_2 \to \tau_3)} \quad (RF_{\triangleright^q})$$

$$\frac{q_1 \leqslant q_2}{q_1(\tau_1 \to \tau_2) \rhd^q q_2\ \mathrm{ref}(\tau_3)} \quad (FR_{\rhd^q})$$

$$\frac{q_1 \leqslant q_2\ \tau_1 \rhd^q \tau_2\ \tau_2 \rhd^q \tau_1}{q_1\ \mathrm{ref}(\tau_1) \rhd^q q_2\ \mathrm{ref}(\tau_2)} \quad (RR_{\rhd^q})$$

$$\frac{q_1 \leqslant q_2\ \tau_3 \rhd^q \tau_1\ \tau_2 \rhd^q \tau_4}{q_1(\tau_1 \to \tau_2) \rhd^q q_2(\tau_3 \to \tau_4)} \quad (FF_{\rhd^q})$$

$\tau_1 \rhd^q \tau_2$ 表示对应的限定词满足限定词之间的部分序关系$\leqslant$。规则($RR_{\rhd q}$)和($FF_{\rhd q}$)类似于限定子类型关系的对应规则。其它的规则都只有最外层限定词作为对应限定词。值得注意的是定义 5 中的规则只是对于值限定词（如 tainted 和 untainted）的一般形式化。对于其它的限定词，这些规则可以根据限定词的性质及分析的需求进行调整。如果 $strip_q(\tau_1) \rhd (strip_q(\tau_2})$（$\tau_1$ 的标准类型是 τ_2 的标准类型的可转化类型）并且 $\tau_1 \rhd^q \tau_2$，那么 τ_1 是 τ_2 的限定可转化类型。限定可转化关系不能保证限定词使用的绝对安全，其原因是 τ_1 是 τ_2 的限定可转化类型且 τ_2 是 τ_3 的限定可转化类型不能保证 τ_1 是 τ_3 的限定可转化类型。

例 3　untainted ref(tainted int) $\rhd^q$ tainted int，且 tainted int $\rhd^q$ tainted ref(untainted int)，但是我们不能得到 untainted ref(tainted int) $\rhd^q$ tainted ref(untainted int)。

3　限定词推断系统

定义 6 展示了源语言的标准类型检查系统。

定义 6　标准的类型检查系统：

$$\frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \quad (Var)$$

$$\frac{}{\Gamma \vdash n : \mathrm{int}} \quad (Int)$$

$$\frac{\Gamma[x \mapsto t] \vdash e : t'}{\Gamma \vdash \lambda x : t.\ e : t \to t'} \quad (Lam)$$

$$\frac{\Gamma \vdash e_1 : t \to t'\ \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 e_2 : t'} \quad (App)$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash \mathrm{ref}(e) : \mathrm{ref}(t)} \quad (Ref)$$

$$\frac{\Gamma \vdash e : \mathrm{ref}(t)}{\Gamma \vdash * e : t} \quad (Deref)$$

$$\frac{\Gamma \vdash e_1 : t\ \Gamma[x \mapsto t] \vdash e_2 : t'}{\Gamma \vdash let\ x = e_1\ \mathrm{in}\ e_2 : t'} \quad (Let)$$

$$\frac{\Gamma \vdash e_1 : \mathrm{ref}(t)\ \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 := e_2 : t} \quad (Assign)$$

$$\frac{\Gamma \vdash e : t'\ t' \rhd t}{\Gamma \vdash (t)e : t} \quad (Cast)$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash annot(e, q_c) : t} \quad (Annot)$$

$\Gamma \vdash e : t$ 表示在类型环境 Γ 中，表达式 e 的类型是 t，其中 Γ 将程序变量映射到它们的标准类型。(Cast)计算出表达式 e 的类型 t'，并检查 t' 为 t 的可转化类型。由于标准类型不含类型限定词，(Annot)忽略限定词的注释。表达式 e 的类型 t 即为该表达式的类型。其它规则的讨论可以参考文献[11]。值得注意的是，由于源语言允许指针类型之间的任意类型转化，静态类型检查不能保证程序没有运行时的错误。

定义 7　限定词推断系统：

$$\frac{x \in dom(\Gamma_q)}{\Gamma_q \vdash_q x : \Gamma_q(x)} \quad (Var_q)$$

$$\frac{q\ \mathrm{fresh}}{\Gamma_q \vdash_q n : q\ \mathrm{int}} \quad (Int_q)$$

$$\frac{\tau = embed_q(t)\quad \Gamma_q[x \mapsto \tau] \vdash_q e : \tau'}{\Gamma_q \vdash_q \lambda x : t.\ e : q(\tau \to \tau')} \quad (Lam_q)$$

$$\frac{\Gamma_q \vdash_q e_1 : q(\tau_1 \to \tau_2)\quad \Gamma_q \vdash_q e_2 : \tau_3\ \tau_3 \leqslant \tau_1}{\Gamma_q \vdash_q e_1 e_2 : \tau_2} \quad (App_q)$$

$$\frac{\Gamma_q \vdash_q e : \tau\ q\ \mathrm{fresh}}{\Gamma_q \vdash_q \mathrm{ref}\ (e) : q\ \mathrm{ref}\ (\tau)} \quad (Ref_q)$$

$$\frac{\Gamma_q \vdash_q e : q\ \mathrm{ref}(\tau)}{\Gamma_q \vdash_q * e : \tau} \quad (Deref_q)$$

$$\frac{\Gamma_q \vdash_q e_1 : \tau\quad \Gamma_q[x \mapsto \tau] \vdash_q e_2 : \tau'}{\Gamma_q \vdash_q \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 : \tau'} \quad (Let_q)$$

$$\frac{\Gamma_q \vdash_q e_1 : q\ \mathrm{ref}(\tau)\quad \Gamma_q \vdash_q e_2 : \tau'\ \tau' \leqslant \tau}{\Gamma_q \vdash_q e_1 := e_2 : \tau'} \quad (Assign_q)$$

$$\frac{\Gamma_q \vdash_q e : \tau\quad \tau' = embed_q(t)\quad \tau \rhd^q \tau'}{\Gamma_q \vdash_q (t)e : \tau'} \quad (Cast_q)$$

$$\frac{\Gamma_q \vdash_q e : q\sigma\quad q = q_c}{\Gamma_q \vdash_q annot(e, q_c) : q\sigma} \quad (Annot_q)$$

在程序通过类型检查的情况下，定义 7 中的限定词推断系统自动推断类型限定词。$\Gamma_q \vdash_q e : \tau$ 表示在限定类型环境 Γ_q 中，表达式 e 的限定类型是 τ，其中 Γ_q 将程序变量映射到它们的限定类型。在规则(Int_q)，(Lam_q)和(Ref_q)中，我们引入新生的限定词变量来表示未知的限定词。在(Lam_q)和($Cast_q$)中，我们使用函数 $embed_q$ 把标准类型映射到嵌入新生限定词变量的限定类型。规则(App_q)和($Assign_q$)生成限定子类型约束：$\tau_1 \leqslant \tau_2$。($Cast_q$)生成限定可转化类型约束：$\tau_1 \rhd^q \tau_2$。($Annot_q$)计算出表达式 e 的限定类型 $q\sigma$ 并生成约束 $q = q_c$（$q \leqslant q_c$ 且 $q_c \leqslant q$），以推断限定词变量 q 的值为 q_c。

定义 8　约束分解规则：

$R(q_1\ \mathrm{int}, q_2\ \mathrm{int}) = \{q_1 \leqslant q_2\}$

$R(q_1\ \mathrm{int}, q_2(\tau_1 \to \tau_2)) = \{q_1 \leqslant q_2\}$

$R(q_1\ \mathrm{int}, q_2\ \mathrm{ref}(\tau)) = \{q_1 \leqslant q_2\}$

$R(q_1(\tau_1 \to \tau_2), q_2\ \mathrm{int}) = \{q_1 \leqslant q_2\}$

$R(q_1\ \mathrm{ref}(\tau), q_2\ \mathrm{int}) = \{q_1 \leqslant q_2\}$

$R(q_1\ \mathrm{ref}(\tau_1), q_2(\tau_2 \to \tau_3)) = \{q_1 \leqslant q_2\}$

$R(q_1\ \mathrm{ref}(\tau_1), q_2\ \mathrm{ref}(\tau_2)) = \{q_1 \leqslant q_2\} \cup R(\tau_1, \tau_2) \cup R(\tau_2, \tau_1)$

$R(q_1(\tau_1 \to \tau_2), q_2\ \mathrm{ref}(\tau_3)) = \{q_1 \leqslant q_2\}$

$R(q_1(\tau_1 \to \tau_2), q_2(\tau_3 \to \tau_4)) = \{q_1 \leqslant q_2\} \cup R(\tau_3, \tau_1) \cup R(\tau_2, \tau_4)$

在程序上执行限定词推断系统后，我们会得到包含 $\tau_1 \leqslant \tau_2$、$\tau_1 \rhd^q \tau_2$ 形式的约束集合 C_τ 和包含 $q = q_c$ 形式的约束集合 C_q。然后，我们通过定义 8 中的规则 $R(\tau, \tau')$ 把 C_τ 规约到 C_q：$C_q = C_q \cup \bigcup_{\forall \tau_1 \leqslant \tau_2 \in C_\tau} R(\tau_1, \tau_2) \cup \bigcup_{\forall \tau_1 \rhd^q \tau_2 \in C_\tau} R(\tau_1, \tau_2)$，其中 $R(\tau, \tau')$ 是定义 5 中规则的从左到右的形式。求解约束集合 C_q 得到限定词变量的值的算法可以参考文献[11]。由于我们假设源程序通过标准类型检查，因此对任意 $\tau_1 \leqslant \tau_2 \in C_\tau$，$strip_q(\tau_1) = strip_q(\tau_2)$（语法等价），且对于任意 $\tau_1 \rhd^q \tau_2 \in C_\tau$，$strip_q(\tau_1) \rhd strip_q(\tau_2)$。

例 4　例 2 中的源程序的限定词推断结果为：

Γ_q：{ $(x, q_2\ \mathrm{ref}(q_1\ \mathrm{int}))$, $(y, q_3\ \mathrm{int})$, $(z, q_5\ \mathrm{ref}(q_4\ \mathrm{int}))$ }

C_τ：{ $q_2\ \mathrm{ref}(q_1\ \mathrm{int}) \rhd^q q_3\ \mathrm{int}$, $q_3\ \mathrm{int} \rhd^q q_5\ \mathrm{ref}(q_4\ \mathrm{int})$ }

C_q：{ $q_1 = \mathrm{tainted}$, $q_4 = \mathrm{untainted}$ }

把 C_τ 规约到 C_q，将得到 C_q = { $q_1 = \mathrm{tainted}$, $q_4 = \mathrm{untainted}$, $q_2 \leqslant q_3$, $q_3 \leqslant q_5$ }。求解 C_q，会得到 q_1 和 q_4 的值分别是 tainted 和 untainted。然而，如例 2 所述，该程序是不安全的，其把 tainted 对象解释成 untainted 对象。但是，根据定义 7 中

的限定词推断系统，我们并没有找到这个安全缺陷，其主要原因是 q_1 的限定词信息并不能传递到 q_4，一些限定词信息在限定词推断的过程中丢失了。第 4 节详细介绍如何解决限定词信息丢失的问题。

4 避免信息丢失

如前所述，定义 7 中的限定词推断系统不足以发现一些含有类型转化的程序的缺陷，其主要原因是限定词推断过程中，一些表达式的类型不足以传递一些有用的限定词信息。因此，在本部分，我们引入变量参与的限定词推断系统，该系统不简单依赖于程序中提供的类型，而是为每个表达式推断一个更适合限定词推断的类型。4.1 节展示变量参与的限定词推断系统；4.2 节引入联合类型来扩展标准类型并给出约束集合的求解过程；4.3 节证明限定词推断的正确性。

4.1 变量参与的推断系统

类似于标准的类型推断系统，我们引入限定类型变量（记作 ρ）来表示需要求解的限定类型。因此，限定类型 τ 变为 $\tilde{\tau}$：

$\tilde{\tau}::=q\tilde{\sigma}$

$\tilde{\sigma}::=\text{int}\mid\text{ref}(\rho)\mid\rho\to\rho$

在推断过程中，为了保留所有类型变量的信息，我们引入限定类型存储（Ψ），将类型变量 ρ 映射到 $\tilde{\tau}$。Γ_ρ 是将程序变量映射为类型变量的类型环境。$\Gamma_\rho,\Psi\vdash_\rho e:\rho,\Psi'$ 表示给定类型环境 Γ_ρ 和类型存储 Ψ，ρ 被推断为表达式 e 的类型变量，且 Ψ 转化成为 Ψ'。定义 10 展示了该推断系统，其规则类似于定义 7 的推断系统的规则。(Lam_ρ) 和 $(Cast_\rho)$ 中的函数 $embed_\varphi$ 映射标准类型到新生的类型变量并在 Ψ 中设置其初始值。定义 9 给出了函数 $embed_\varphi$ 的具体定义。

定义 9 函数 $embed_\varphi$ 的定义为：

$embed_\varphi(\text{int},\Psi)=(\rho,\Psi\cup\{(\rho,q\ \text{int})\})\quad\rho,q\ \text{fresh}$

$embed_\varphi(\text{ref}(t),\Psi)=(\rho,\Psi'\cup\{(\rho,q\ \text{ref}(\rho'))\})\quad\rho,q\ \text{fresh}$

其中 $(\rho',\Psi')=embed_\varphi(t,\Psi)$。

$embed_\varphi(t_1\to t_2,\Psi)=(\rho,\Psi''\cup\{(\rho,q\ (\rho'\to\rho''))\})\quad\rho,q\ \text{fresh}$

其中 $(\rho',\Psi')=embed_\varphi(t_1,\Psi),(\rho'',\Psi'')=embed_\varphi(t_2,\Psi')$。

定义 10 变量参与的限定词推断系统：

$$\frac{x\in dom(\Gamma_\rho)}{\Gamma_\rho,\Psi\vdash_\rho x:\Gamma_\rho(x),\Psi}\quad(Var_\rho)$$

$$\frac{\rho,q\ \text{fresh}}{\Gamma_\rho,\Psi\vdash_\rho n:\rho,\Psi\cup\{(\rho,q\ \text{int})\}}\quad(Int_\rho)$$

$$\frac{embed_\varphi(t,\Psi)=(\rho,\Psi')\quad\Gamma_\rho[x\mapsto\rho],\Psi'\vdash_\rho e:\rho',\Psi''}{\Gamma_\rho,\Psi\vdash_\rho\lambda x:t.\,e:\rho'',\Psi''\cup\{(\rho'',q(\rho\to\rho'))\}}\quad(Lam_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e:\rho,\Psi'\quad(\rho,q\ \text{ref}(\rho'))\in\Psi'}{\Gamma_\rho,\Psi\vdash_\rho *e:\rho',\Psi'}\quad(Deref_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e:\rho,\Psi'\quad\rho',q\ \text{fresh}}{\Gamma_\rho,\Psi\vdash_\rho\text{ref}(e):\rho',\Psi'\cup\{(\rho',q\ \text{ref}(\rho))\}}\quad(Ref_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e_1:\rho,\Psi'\quad\Gamma_\rho[x\mapsto\rho],\Psi'\vdash_\rho e_2:\rho',\Psi''}{\Gamma_\rho,\Psi\vdash_\rho\text{let}\ x=e_1\ \text{in}\ e_2:\rho',\Psi''}\quad(Let_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e_1:\rho_1,\Psi'\quad\Gamma_\rho,\Psi'\vdash_\rho e_2:\rho_2,\Psi''\quad(\rho_1,q(\rho_3\to\rho_4))\in\Psi''\quad\rho_2\leqslant\rho_3}{\Gamma_\rho,\Psi\vdash_\rho e_1e_2:\rho_4,\Psi''}\quad(App_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e_1:\rho,\Psi'\quad\Gamma_\rho,\Psi'\vdash_\rho e_2:\rho',\Psi''\quad(\rho,q\ \text{ref}(\rho''))\in\Psi''\quad\rho'\leqslant\rho''}{\Gamma_\rho\vdash_\rho e_1:=e_2:\rho',\Psi''}\quad(Assign_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e:\rho,\Psi'\quad embed_\varphi(t,\Psi')=(\rho',\Psi'')\quad\rho\leqslant\rho'}{\Gamma_\rho,\Psi\vdash_\rho(t)e:\rho',\Psi''}\quad(Cast_\rho)$$

$$\frac{\Gamma_\rho,\Psi\vdash_\rho e:\rho,\Psi'\quad(\rho,q\tilde{\sigma})\in\Psi'\quad q=q_c}{\Gamma_\rho,\Psi\vdash_\rho annot(e,q_c):\rho,\Psi'}\quad(Annot_\rho)$$

现在，我们简单讨论定义 10 中的规则：

- (Var_ρ)：从类型环境 Γ_ρ 中得到变量 x 的类型变量并保持类型存储 Ψ 不变。
- (Int_ρ)：赋予整数 n 新生类型变量 ρ，并在类型存储中设置其初值为 q int，其中 q 是新生的限定词变量。
- (Lam_ρ)：首先，由 $embed_\varphi(t,\Psi)$ 得到 t 的类型变量 ρ 及类型存储 Ψ'。其次，假设参数 x 的类型变量为 ρ，类型存储为 Ψ'，计算函数体的类型变量为 ρ' 并得到新的类型存储 Ψ''。最后，赋予函数新类型变量 ρ'' 并在 Ψ'' 中设置其初始值为 $q(\rho\to\rho')$。
- $(Deref_\rho)$：计算表达式 e 的类型变量为 ρ 并从类型存储中取得 ρ 的值为 q ref(ρ')。$*e$ 的类型变量即为 ρ'。
- (Ref_ρ)：计算表达式 e 的类型变量为 ρ 并在类型存储中设置新生类型变量 ρ' 的值为 q ref(ρ)。ρ' 即为表达式 ref(e) 的类型变量。
- (Let_ρ)：首先计算 e_1 的类型变量为 ρ，然后假设 x 的类型变量为 ρ，计算 e_2 的类型变量为 ρ'。
- (App_ρ)：首先计算 e_1 的类型变量为 ρ_1，e_2 的类型变量为 ρ_2，然后从类型存储中取得 ρ_1 的值为 $q(\rho_3\to\rho_4)$ 并生成约束 $\rho_2\leqslant\rho_3$。
- $(Assign_\rho)$：首先计算 e_1 的类型变量为 ρ，e_2 的类型变量为 ρ'，然后从类型存储中取得 ρ 的值为 q ref(ρ'') 并生成约束 $\rho'\leqslant\rho''$。
- $(Cast_\rho)$：首先计算 e 的类型变量为 ρ 并由函数 $embed_\varphi$ 得到 t 的类型变量 ρ'，然后生成约束 $\rho\leqslant\rho'$。
- $(Annot_\rho)$：首先计算表达式 e 的类型变量为 ρ 并从类型存储中取得 ρ 的值为 $q\tilde{\sigma}$，然后生成约束 $q=q_c$。

总之，当引入新类型变量的时候，我们都在类型存储中设置其初始值（参见 (Int_ρ)，(Lam_ρ) 和 (Ref_ρ)）。规则 (App_ρ)，$(Assign_\rho)$ 和 $(Cast_\rho)$ 在类型变量上生成 $\rho\leqslant\rho'$ 形式的约束。在下一节中，我们通过这些约束为每个表达式求解出适合进行限定词推断的类型。

当程序通过该限定词推断系统后，我们得到一个类型存储 Ψ，一个包含 $\rho\leqslant\rho'$ 形式的约束集合 C_ρ 和一个包含 $q=q_c$ 形式的约束集合 C_q。值得注意的是，通过一个简单的替换过程将类型变量映射到它们的限定类型，定义 10 中的限定词推断系统和定义 7 中的系统是等价的。

例 5 对于该系统，例 2 中源程序的推断结果为：

$\Gamma_q:\{(x,\rho_2),(y,\rho_3),(z,\rho_5)\}$

$\Psi:\{(\rho_1,q_1\ \text{int}),(\rho_2,q_2\ \text{ref}(\rho_1)),(\rho_3,q_3\ \text{int}),(\rho_4,q_4\ \text{int}),(\rho_5,q_5\ \text{ref}(\rho_4))\}$

$C_\rho:\{\rho_2\leqslant\rho_3,\rho_3\leqslant\rho_5\}$

$C_q:\{q_1=\text{tainted},q_4=\text{untainted}\}$

4.2 扩展和求解

在本节中，我们利用定义 10 中的推断系统产生的约束集合 C_ρ、C_q 以及类型存储 Ψ 来求解每个表达式的适合进行限定词推断的类型。我们首先在 4.2.1 节中引入扩展类型，然后在 4.2.2 节中给出具体的求解过程。

4.2.1 扩展类型

考虑例1中的程序，对于程序变量 a 和 b，类型 int * 可比 int 传递更多有用的限定词信息，因此，int * 比 int 更适合进行限定词的推断。但是，我们也会遇到其它的情况，如变量 a 的类型是 int(*)(int)(函数指针类型)，变量 x 的类型是 int * *。通过类型转化，变量 x 的值可以赋值给 a。那么，在这种情况下，对于变量 a，什么类型最适合进行限定词的推断？显然，类型 int(*)(int)和 int * * 都不能包含所有的限定词信息，因此最好的解决办法便是引入联合类型。

我们通过引入联合类型来扩展标准类型：

$t ::= \text{int} \mid \text{ref}(t) \mid t \rightarrow t \mid \text{ref}(t) \vee t \rightarrow t$

此时，扩展的限定类型的定义为：

$\tau ::= q_c\sigma, q_c \in Q$

$\sigma ::= \text{int} \mid \text{ref}(\tau) \mid \tau \rightarrow \tau \mid \text{ref}(\tau) \vee \tau \rightarrow \tau$

定义11定义了 t 的子类型关系($\prec$)。如果将类型对应到拥有该类型的值的集合，则子类型关系对应于集合的包含关系。关于联合类型的子类型关系的进一步讨论可以参考文献[4]。扩展的限定类型 τ 的限定子类型关系可以通过子类型关系很容易得到，为节省空间，我们省略了其定义。此外，为阐述方便，规定所有操作符的优先级顺序为：ref $>\!\!\rightarrow\!\!>$ $\vee$ $>\!\!<\!\!>$ $\sqcup$。

定义11 扩展类型的子类型关系($\prec$)：

$$\frac{}{\text{int} \prec \text{int}} \qquad \frac{t_1 \prec t_2}{\text{ref}(t_1) \prec \text{ref}(t_2)}$$

$$\frac{t_3 \prec t_1 \quad t_2 \prec t_4}{t_1 \rightarrow t_2 \prec t_3 \rightarrow t_4} \qquad \frac{\text{ref}(t_1) \prec \text{ref}(t_2)}{\text{ref}(t_1) \prec \text{ref}(t_2) \vee t_3 \rightarrow t_4}$$

$$\frac{t_1 \rightarrow t_2 \prec t_4 \rightarrow t_5}{t_1 \rightarrow t_2 \prec \text{ref}(t_3) \vee t_4 \rightarrow t_5} \qquad \frac{\text{ref}(t_1) \prec \text{ref}(t_4) \quad t_2 \rightarrow t_3 \prec t_5 \rightarrow t_6}{\text{ref}(t_1) \vee t_2 \rightarrow t_3 \prec \text{ref}(t_4) \vee t_5 \rightarrow t_6}$$

给定任意两个类型 t_1 和 t_2，定义12给出计算可以包含它们的所有限定词信息的类型的规则。在程序中，如果 t_1 类型的变量实际上被赋予 t_2 类型的值，那么，$t_1 \sqcup t_2$ 是最适合进行限定词推断的类型。为简明起见，定义12只给出了部分规则，对于任意 t_1 和 t_2，$t_1 \sqcup t_2 = t_2 \sqcup t_1$。

定义12 计算规则：

$\text{int} \sqcup t = t$

$\text{ref}(t_1) \sqcup \text{ref}(t_2) = \text{ref}(t_1 \sqcup t_2)$

$\text{ref}(t_1) \sqcup t_2 \rightarrow t_3 = \text{ref}(t_1) \vee t_2 \rightarrow t_3$

$t_1 \rightarrow t_2 \sqcup t_3 \rightarrow t_4 = (t_1 \sqcup t_3) \rightarrow (t_2 \sqcup t_4)$

$\text{ref}(t_1) \sqcup \text{ref}(t_2) \vee t_3 \rightarrow t_4 = (\text{ref}(t_1) \sqcup \text{ref}(t_2)) \vee t_3 \rightarrow t_4$

$t_1 \rightarrow t_2 \sqcup \text{ref}(t_3) \vee t_4 \rightarrow t_5 = \text{ref}(t_3) \vee (t_1 \rightarrow t_2 \sqcup t_4 \rightarrow t_5)$

$\text{ref}(t_1) \vee t_2 \rightarrow t_3 \sqcup \text{ref}(t_4) \vee t_5 \rightarrow t_6 = (\text{ref}(t_1) \sqcup \text{ref}(t_4)) \vee (t_2 \rightarrow t_3 \sqcup t_5 \rightarrow t_6)$

例6 $\text{int} \sqcup \text{ref}(\text{int}) = \text{ref}(\text{int})$，$\text{ref}(\text{ref}(\text{int})) \sqcup \text{ref}(\text{int} \rightarrow \text{int}) = \text{ref}(\text{ref}(\text{int}) \vee \text{int} \rightarrow \text{int})$。

4.2.2 求解过程

根据定义10中的推断系统，我们得到类型存储 Ψ 以及约束集合 C_ρ 和 C_q。在本节中，将逐步介绍具体的求解过程。

(1)根据 Ψ，通过一个简单的替换过程，可得到一个初始解 S'，其将每个类型变元 ρ 映射到其初始类型 t。该初始类型为对应表达式的标准类型。

例7 例5中的推断结果的初始解为：

$S' = \{(\rho_1, \text{int}), (\rho_2, \text{ref}(\text{int})), (\rho_3, \text{int}), (\rho_4, \text{int}), (\rho_5, \text{ref}(\text{int}))\}$，其中，$\rho_2$，$\rho_3$ 和 ρ_5 的初始类型分别为变量 x，y 和 z 的标准类型。

(2)由解 S'，我们需要得到满足下面性质的解 S：

① $\forall \rho \in \text{dom}(\Psi), \exists t, S'(\rho) \sqcup t = S(\rho)$，其中 $\text{dom}(\Psi)$ 为 Ψ 的定义域。该性质保证我们不会丢失限定词变量上的基本约束。

② $\forall \rho \in \text{dom}(\Psi), \exists t, \Psi(\rho)(S) \sqcup t = S(\rho)$，$\Psi(\rho)(S)$ 指用 $S(\rho')$ 替换 $\Psi(\rho)$ 中的任意类型变量 ρ' 并去掉其最外层限定词变量后得到的类型。该性质保证保留类型变量之间的原始关系。

为了得到解 S，我们把 Ψ 和 C_ρ 转化成等式系统 E：对任意 $\rho \in \text{dom}(\Psi)$，构造等式 $\rho = \tilde{\sigma} \sqcup \rho_1 \sqcup \cdots \sqcup \rho_n$，其中 $\Psi(\rho) = q\tilde{\sigma}$，且 $\forall \rho', \rho' \in \{\rho_1 \cdots \rho_n\}$ 当且仅当 $\rho' \leqslant \rho \in C_\rho$。

例8 根据例5和例7中的结果，等式系统 E 为：$\rho_1 = \text{int}$，$\rho_2 = \text{ref}(\rho_1)$，$\rho_3 = \text{int} \sqcup \rho_2$，$\rho_4 = \text{int}$，$\rho_5 = \text{ref}(\rho_4) \sqcup \rho_3$。

(3)运用 Courcelle 在无限树上的工作[7]，对等式系统进行求解。算法1给出了递归模式的算法，该算法的主要思想是通过不断替换来消元。Computing(R_ρ, S)根据定义12中的计算规则来求解 R_ρ 并得到类型 t，在此过程中，我们用 $S(\rho')$ 来替换 R_ρ 中的任意类型变量 ρ'。

算法1 Equation_Solve

输入：等式系统 E，初始解 S′

输出：S

```
1. if E=∅ then
2.    return S′;
3. else
4.    从 E 中任取 ρ=R_ρ;
5.    let E′={ρ′= (R_ρ′{R_ρ/ρ}) | (ρ′= R_ρ′)∈E/(ρ=R_ρ) };
6.    let S=Equation_Solve(E′,S′);
7.    let t=Computing(R_ρ,S);
8.    S(ρ) := t;
9. return S
```

例9 给定例7中的初始解 S' 和例8中的等式系统 E，解 S 为：$S = \{(\rho_1, \text{int}), (\rho_2, \text{ref}(\text{int})), (\rho_3, \text{ref}(\text{int})), (\rho_4, \text{int}), (\rho_5, \text{ref}(\text{int}))\}$

(4)现在我们得到了解 S，其映射每个类型变量 ρ 到其适合进行限定词推断的类型。接下来的问题便是在限定词推断过程中如何利用 S。算法2给出计算每个类型变量的最终的限定类型的算法。$Ufy(t, \tau)$ 从外至内把 τ 的限定词变量嵌入类型 t，且对于 t 中多余的限定词位置，使用函数 $embed_q$ 嵌入新生的限定词变量。定义13给出了具体的定义。值得注意的是，对于算法2，根据关系($\prec$)，我们要求 Ψ 满足从小到大的顺序，其中，$(\rho, \tilde{\tau}) \prec (\rho', \tilde{\tau}')$ 当且仅当 ρ 出现于 $\tilde{\tau}'$。

算法2 Embed_Qualifier_Variables

输入：S，有序的 Ψ

输出：S_q

```
1. let S_q=∅;
2. for i=1 to |Ψ| do
3.    let (ρ,τ̃)=Ψ[i];
4.    let t=S(ρ);
5.    switch(τ̃) {
6.       case q int:S_q= S_q∪{(ρ,Ufy(t,q int))};
7.          break;
8.       case q ref(ρ1):S_q= S_q∪{(ρ,Ufy(t,q ref(S_q(ρ1))))};
9.          break;
```

10. case $q(\rho_1 \to \rho_2)$: $S_q = S_q \cup \{(\rho, Ufy(t, q(S_q(\rho_1) \to S_q(\rho_2))))\};\}$

11. return S_q;

定义 13 函数 Ufy 的定义为：

$Ufy(\text{int}, q\ \text{int}) = q\ \text{int}$

$Ufy(\text{ref}(t), q\ \text{int}) = q\ \text{ref}(embed_q(t))$

$Ufy(\text{ref}(t), q\ \text{ref}(\tau)) = q\ \text{ref}(Ufy(t, \tau))$

$Ufy(t_1 \to t_2, q\ \text{int}) = q\ (embed_q(t_1) \to embed_q(t_2))$

$Ufy(t_1 \to t_2, q\ (\tau_1 \to \tau_2)) = q\ (Ufy(t_1, \tau_1) \to Ufy(t_2, \tau_2))$

$Ufy(\text{ref}(t_1) \vee t_2 \to t_3, q\ \text{int}) = q\ (\text{ref}(embed_q(t_1)) \vee embed_q(t_2) \to embed_q(t_3))$

$Ufy(\text{ref}(t_1) \vee t_2 \to t_3, q\ \text{ref}(\tau)) = q\ (\text{ref}(Ufy(t_1, \tau)) \vee embed_q(t_2) \to embed_q(t_3))$

$Ufy(\text{ref}(t_1) \vee t_2 \to t_3, q\ (\tau_1 \to \tau_2)) = q\ (\text{ref}(embed_q(t_1)) \vee Ufy(t_2, \tau_1) \to Ufy(t_3, \tau_2))$

$Ufy(\text{ref}(t_1) \vee t_2 \to t_3, q\ (\text{ref}(\tau_1) \vee \tau_2 \to \tau_3)) = q\ (\text{ref}(Ufy(t_1, \tau_1)) \vee Ufy(t_2, \tau_2) \to Ufy(t_3, \tau_3))$

例 10 根据例 5 中的 Ψ 和例 9 中的 S，我们得到 S_q 为：$S_q = \{(\rho_1, q_1\ \text{int}), (\rho_2, q_2\ \text{ref}(q_1\ \text{int})), (\rho_3, q_3\ \text{ref}(q_6\ \text{int})), (\rho_4, q_4\ \text{int}), (\rho_5, q_5\ \text{ref}(q_4\ \text{int}))\}$，其中 q_6 为新生的限定词变量。

(5)根据算法 2 的结果 S_q，将 C_ρ 规约到 C_q：$C_q = C_q \cup \bigcup_{\forall \rho_1 \leqslant \rho_2 \in C_\rho} R(S_q(\rho_1), S_q(\rho_2))$，其中，$R(\tau_1, \tau_2)$ 是定义 8 的简单扩展，为节省空间，我们省略了新的定义。求解 C_q 的算法可以参考文献[11]。

例 11 根据例 5 和例 10，我们得到新的限定词变量的约束集合：$C_q = \{q_1 = \text{tainted}, q_4 = \text{untainted}, q_2 \leqslant q_3, q_1 = q_6, q_3 \leqslant q_5, q_4 = q_6\}$。对比例 4 中的结果，$C_q$ 包含了另外两个约束即 $q_1 = q_6$ 和 $q_4 = q_6$，意为我们通过一个新生的限定词变量 q_6 在 q_1 和 q_4 之间传递限定词信息。显然，该 C_q 是不可满足的(tainted≠untainted)，即我们发现了例 2 中程序的缺陷。

4.3 正确性证明

由于源语言允许指针类型之间的任意类型转化，静态的类型检查不能保证程序没有运行时的错误，因此本节就限定词推断来证明正确性，即，通过定理 1 证明类型的扩展及约束求解不会导致限定词变量的不正确约束。

定义 14 $\forall t_1, t_2, t_1 \rightarrowtail t_2$ 等价于 $\exists t, t_1 \sqcup t = t_2$。

定义 15 $\forall q, \tau_1, \tau_2, q \in \tau_1$ 表示 q 是 τ_1 的限定词变量，$q \in \tau_1 \uplus \tau_2$ 表示 q 是 τ_1 或 τ_2 的限定词变量。

引理 1 $\forall\ \tau_1, \tau_2, \tau_3, \forall q_1, q_2 \in \tau_1 \uplus \tau_3$ 满足下面的性质：

如果 $R(\tau_1, \tau_2) \cup R(\tau_2, \tau_3) \Rightarrow q_1 \leqslant q_2$，则 $q_1 \leqslant q_2 \in R(\tau_1, \tau_3)$。

证明：在类型的结构上进行归纳证明。

引理 2 $\forall \tau_1, \tau_2, t_3, t_4$，令 $t_1 = strip_q(\tau_1), t_2 = strip_q(\tau_2), \tau_3 = Ufy(t_3, \tau_1), \tau_4 = Ufy(t_4, \tau_2)$，如果 $t_1 \rightarrowtail t_3, t_2 \rightarrowtail t_4$，则 $\forall q_1, q_2 \in \tau_1 \uplus \tau_2$ 满足下面的性质：

$q_1 \leqslant q_2 \in R(\tau_3, \tau_4)$ 当且仅当 $q_1 \leqslant q_2 \in R(\tau_1, \tau_2)$。

证明：在类型的结构上进行归纳证明。

定理 1(正确性) 给定 Ψ, C_ρ 和 S_q，对 $\forall \rho_1, \rho_n \in \text{dom}(\Psi)$，如果 $\exists \rho_1 \leqslant \rho_2 \leqslant \cdots \leqslant \rho_n \in C_\rho$，令 $\Phi = R(S_q(\rho_1), S_q(\rho_2)) \cup \cdots \cup R(S_q(\rho_{n-1}), S_q(\rho_n))$，则下面的性质成立：

① $\forall q_1, q_2 \in S_q(\rho_1) \uplus S_q(\rho_n)$，如果 $\Phi \Rightarrow q_1 \leqslant q_2$，则 $q_1 \leqslant q_2 \in R(S_q(\rho_1), S_q(\rho_n))$。

②令 τ_1, τ_n 分别为 ρ_1, ρ_n 的初始限定类型，$\forall q_1, q_2 \in \tau_1 \uplus \tau_n$，如果 $\Phi \Rightarrow q_1 \leqslant q_2$，则 $q_1 \leqslant q_2 \in R(\tau_1, \tau_n)$，其中初始限定类型可通过一个简单的替换过程，由 Ψ 直接得到。

证明：根据引理 1，在 ρ_1 到 ρ_n 的路径长度上进行归纳可证明第一个性质。然后根据第一个性质和引理 2，可证明第二个性质。

5 实例分析

我们用 Ocaml 实现了源语言的限定词推断系统并对 15 个含有多种类型转化的程序进行了测试。在本节中，我们首先展示一些来源于测试程序的代表性的两层的类型转化和它们的推断结果，然后详细介绍两个较复杂的测试程序的推断结果。

为了简明，例 12 的类型转化列表只在标准类型中标注了关键的限定词和限定词变量。未标注位置的限定词变量的值被推断为 tainted 和 untainted，即它们的值没有被约束。⇒表示类型转化，R 表示推断结果。通过该列表，可以发现我们的系统在处理这些类型转化的时候既没有丢失限定词信息也没有引入不必要的约束。

例 12 测试的典型的两层类型转化的列表为：

- ref(tainted int)⇒int⇒ref(q_1 int)
  R={q_1 = tainted }
- ref((tainted int)→int)⇒int⇒ref((q_1 int)→int)
  R={q_1 = tainted }
- ref((tainted int)→int)⇒ref(ref(int))⇒ref((q_1 int)→int)
  R={q_1 = tainted }
- ref(tainted int)⇒ref(q_1(int→int))⇒ref(q_2 int)
  R={q_1 = tainted, q_2 = tainted}
- ref(ref(tainted int))⇒ref(int→int)⇒ref(ref(q_1 int))
  R={q_1 = tainted }
- ref(ref(tainted int)→ref(int))⇒ref(int→int)⇒ref(ref(q_1 int) →ref(int))
  R={q_1 = tainted }

例 13 测试程序 1：

```
let y=ref λx:int. annot(x,untainted) in
let z=(ref(ref(int)) y in
let w=(int) z in
let g=(ref(int)) w in
let h=(ref(int→int)) g in
   (*h) annot(3,tainted)
```

例 14 测试程序 2：

```
let y=ref λx:int. annot(x,untainted) in
let z=(ref(ref(int)) y in
let w=(int) *z in
let g=(ref(int)) w in
let h=(ref(int→int)) g in
   (*h) annot(3,tainted)
```

例 13 和例 14 中的程序都含有复杂的类型转化。它们唯一的区别在于第三行：分别将 z 和 $*z$ 的值进行类型转化并赋值于 w。因此，在程序 1 中，$*h$ 和 $*y$ 表示同一个函数且

该函数要求 untainted 数据作为参数。但是最后一行将 tainted 数据通过 $*h$ 传递给该函数。对于该程序，我们的推断系统发现了这个安全缺陷并报告限定词使用错误。程序 2 的最后一行有运行时间的解引用错误。由于类型系统的局限性，该错误不能通过类型检查系统发现。但就限定词而言，由于变量 h 和 y 有不同的值，即 h 不指向要求 untainted 数据作为参数的函数，该程序是安全的。对于该程序，我们的推断系统推断其为安全的程序并推断出 h 指向接收 tainted 数据作为参数的函数。根据上述测试程序，我们的限定词推断系统可以正确处理含有复杂类型转化的程序。

6 相关工作

限定类型可以看成是一种特殊的精化类型[2,13]。限定类型扩展类型系统的表达能力但不改变类型的基本结构。文献[9]中提出的类型限定词理论给出了支持用户定义类型限定词和在程序中添加限定词注释的架构以及流不敏感的限定词推断系统。文献[10]中提出了流敏感的限定词推断系统，其中只有限定词是流敏感的，基本的类型仍然是流不敏感的。文献[16]中的类型精化的架构支持流敏感的复杂的类型系统。在文献[6]中，允许用户明确定义类型限定词的语义及推导规则的架构被提出，该架构目前仍然不够灵活，只能支持特定种类的类型限定词。基于类型限定词的理论，CQual[8]被开发并用于 C 程序的静态分析。目前，CQual 已有很多的应用，如常量限定词(const)的推断[11]，查找格式化字符串缺陷[17]，授权钩放置的静态分析[19,12]，查找用户/内核漏洞[15]，以及 Linux 内核死锁问题的检测[1]等。之后，JQual[14]被开发并用于 Java 程序的静态分析。

本文提出基于联合类型和类型约束求解的流不敏感的限定词推断。文献[20]提出基于类型推断的指针指向分析，其引入的非标准类型类似于本文引入的联合类型，但其类型推断过程中的约束条件比本文更强，这对于限定词推断是不必要的。文献[21]提出基于类型推断的内容敏感的流分析。

结束语 基于类型限定词的程序分析是一种用于提高软件质量的静态分析技术。然而，由于 C 标准允许指针之间的任意类型转化以及其它一些类型转化[5,18]，标准的限定词的推断过程会丢失一些有用的限定词信息。本文旨在解决流不敏感的限定词推断过程中限定词信息丢失的问题。本文首先在一个允许类型转化的源语言上形式化了流不敏感的限定词推断系统，然后引入联合类型并为每个表达式推断更适合进行限定词推断的类型，从而有效避免了限定词推断过程中信息的丢失。

参 考 文 献

[1] Aiken A, Foster J S, Kodumal J, et al. Checking and inferring local non-aliasing[J]. ACM SIGPLAN Notices, ACM, 2003, 38(5): 129-140

[2] Bierman G M, Gordon A D, Hriţcu C, et al. Semantic subtyping with an SMT solver[J]. ACM SIGPLAN Notices, ACM, 2010, 45(9): 105-116

[3] Broadwell P, Harren M, Sastry N. Scrash: A system for generating secure crash information[C]// Proceedings of the 12th conference on USENIX Security Symposium. Volume 12, USENIX Association, 2003: 19-19

[4] Castagna G, Xu Zhi-wu. Set-theoretic foundation of parametric polymorphism and subtyping[J]. ACM SIGPLAN Notices, ACM, 2011, 46(9): 94-106

[5] Chandra S, Reps T. Physical type checking for C[J]. ACM SIGSOFT Software Engineering Notes, ACM, 1999, 24(5): 66-75

[6] Chin B, Markstrum S, Millstein T. Semantic type qualifiers[J]. ACM SIGPLAN Notices, ACM, 2005, 40(6): 85-95

[7] Courcelle B. Fundamental properties of infinite trees[J]. Theoretical computer science, 1983, 25(2): 95-169

[8] Foster J S. CQUAL User's Guide Version 0.991[K]. 2004

[9] Foster J S, Fähndrich M, Aiken A. A theory of type qualifiers[J]. ACM SIGPLAN Notices, ACM, 1999, 34(5): 192-203

[10] Foster J S, Terauchi T, Aiken A. Flow-sensitive type qualifiers[M]. ACM, 2002

[11] Foster J S. Type qualifiers: lightweight specifications to improve software quality[D]. Berkeley: University of California, 2002

[12] Fraser T, Petroni Jr N L, Arbaugh W A. Applying flow-sensitive CQUAL to verify MINIX authorization check placement[C]// Proceedings of the 2006 workshop on Programming languages and analysis for security. ACM, 2006: 3-6

[13] Freeman T, Pfenning F. Refinement types for ML[M]. ACM, 1991

[14] Greenfieldboyce D, Foster J S. Type qualifier inference for Java[J]. ACM SIGPLAN Notices, ACM, 2007, 42(10): 321-336

[15] Johnson R, Wagner D. Finding User/Kernel Pointer Bugs with Type Inference[J]. USENIX Security Symposium, 2004, 2(0): 0

[16] Mandelbaum Y, Walker D, Harper R. An effective theory of type refinements[J]. ACM SIGPLAN Notices, ACM, 2003, 38(9): 213-225

[17] Shankar U, Talwar K, Foster J S, et al. Detecting Format String Vulnerabilities with Type Qualifiers[C]// USENIX Security Symposium. 2001: 201-220

[18] Siff M, Chandra S, Ball T, et al. Coping with type casts in C[C]// Software Engineering—ESEC/FSE'99. Springer Berlin Heidelberg, 1999: 180-198

[19] Zhang Xiao-lan, Edwards A, Jaeger T. Using CQUAL for Static Analysis of Authorization Hook Placement[C]// USENIX Security Symposium. 2002: 33-48

[20] Steensgaard B. Points-to analysis in almost linear time[C]// Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on principles of programming languages. ACM, 1996: 32-41

[21] Fähndrich M, Rehof J, Das M. Scalable context-sensitive flow analysis using instantiation constraints[J]. ACM SIGPLAN Notices, 2000, 35(5): 253-263