arXiv:2603.22032v1 [cs.PL] 23 Mar 2026

Erlang 的集合论类型：理论、实现与评估

阿尔伯特·席姆普夫
凯撒斯劳滕德国应用科学大学（邮箱：albert.schimpf@rptu.de）

STEFAN WEHR
奥芬堡应用科学大学，德国（邮箱：stefan.wehr@hs-offenburg.de）

ANNETTE BIENIUSA
凯撒斯劳滕 RPTU 德国大学（电子邮箱：bieniusa@rptu.de）

摘要

Erlang 的动态类型制度可能导致运行时错误，即使进程重启后，这些错误仍可能持续存在。其中一些运行时错误可以通过静态类型检查来避免。尽管 Erlang 提供了类型规范语言，但编译器并不强制执行这些类型，因此它们的作用仅限于文档说明。由于该语言具有动态类型测试、子类型关系、等递归类型、多态、签名中的交类型以及无标签联合类型等特性，检查 Erlang 代码的类型颇具挑战。

本文提出了一种适用于 Erlang 的集合论类型系统，涵盖了 Erlang 现有类型语言的核心特性。该形式化类型系统保证类型可靠性，并确保类型检查仍然可判定。此外，本文还提供了一个类型检查器的实现，支持 Erlang 类型语言的所有特性以及大多数顶级语言构造。通过对 Erlang 标准库中的模块、一个外部项目以及类型检查器自身进行案例研究，展示了该类型检查器在验证真实世界的 Erlang 代码方面的有效性。

1 引言

Erlang 是一种函数式语言，旨在用于高度并发的分布式系统，这类系统能够提供软实时保证并具备故障恢复能力。自20世纪80年代问世以来， [2], 它已广泛应用于各个行业，包括电信、消息传递、金融和游戏。

在生产环境中，安全地处理错误至关重要。Erlang 最常见且受到推崇的做法是让故障进程崩溃并重启。虽然这种方法对许多故障都有效，但有些错误（例如类型不匹配）无论重启多少次都会持续存在。静态类型系统能在执行前防止此类问题 [37], , 但 Erlang 采用动态类型，只能在运行时检测到这些问题。尽管 Erlang 提供了用于函数签名的类型规范语言 [21, 27],, 编译器却不会强制执行这些规范，因此它们只能充当（有时并不准确的）文档，而无法用于验证。

由于动态类型测试、子类型关系、等递归类型 [1, 17], 多态、签名中的交类型以及无标签联合类型 [36]等特性，在 Erlang 中进行静态类型检查颇具挑战。已有多项研究探索了在 Erlang 中引入静态类型检查 [24, 31, 32, 39, 41, 46, 49]。传统子类型关系由句法推导规则定义，

通常难以处理交类型和并集类型，因为分配律并非总是成立，而且最小上界/最大下界通常无法表示 [19, 33, 36]。

语义子类型 [10, 11, 14, 23] 提供了一种替代方法：将类型解释为集合，从而将子类型关系归约为集合包含。这种方法自然支持交集、并集和否定类型，但需要精心设计的类型模型和复杂的子类型算法。

本文提出了一种用于 Erlang 的集合论类型系统，并将其形式化为声明式系统和算法式系统。与最先进的 Erlang 类型检查器 [44, 49] 和静态分析器 [41], 不同，我们的系统以可靠且富有表达力的理论为基础，支持广泛的语言特性，唯一值得注意的例外是非正则多态类型。

我们的工作在理论和实践两方面都建立在先前发表的成果 [42] 之上，并对其加以拓展。在理论方面，我们正式确立了类型可靠性，并证明了类型检查的可判定性。在实践方面，我们实现了一个原型类型检查器，能够处理 Erlang 的多种语言特性，并通过一个真实案例研究展示了其实用性。

贡献。 我们提出以下贡献：

- 我们基于集合论类型，为 Erlang 形式化了一种类型系统（第 3节）。
- 我们确立了该类型系统的可靠性（第 3.4节和第 A节），并证明在我们的形式体系中，类型检查是可判定的（第 3.5 节和第 B节）。
- 我们使用 Erlang 开发并实现了 Etylizer 类型检查器。Etylizer 支持 Erlang 类型语言的所有特性，以及项级语言构造的一个全面子集，使其能够应用于真实项目（第 4节）。
- 我们通过将 Etylizer 应用于 Erlang 标准库中的模块（第 4.2节）、一个第三方自由及开源软件项目以及 Etylizer 自身的代码库（第 4.3节），对其进行了实证评估。
- 借助 Etylizer，我们首次实现了一个完全独立的集合论类型系统；该实现独立于 CDuce 的实现，并非由其衍生而来 [15]。

关于可复现性。此前关于集合论类型的研究均由同一个研究团队开展，现有的支持余递归类型的实现也都基于 CDuce 框架 [15]。据我们所知，数十年来，我们首次从理论和实践两方面验证了这项先前研究。我们还独立开发了一个完整的类型检查器，集成了真实代码库所需的所有功能，为该领域的后续研究奠定了新的基础，使整体价值超越各部分之和。

路线图。第2 节通过示例介绍我们的类型系统。第3节介绍形式化类型系统和类型推导算法，以及相关的元理论结果。第4节讨论实现和一个真实世界的案例研究。第5节介绍相关工作，第6 节给出结论。所有证明见第A 节和第B节，而第C 节和第D节则展示了关于在真实世界的 Erlang 代码中使用类型测试和值依赖类型的实证评估结果。

2 Erlang 与集合论类型入门

为介绍集合论类型的原则，我们首先讨论 Erlang（类型）语言中若干难以用基于规则的子类型系统处理、但非常适合集合论类型和语义子类型的方面。此次讨论以先前一篇文章中呈现的示例为基础并加以扩展 [42], , 为理解集合论类型在 Erlang 场景中的优势和应用奠定基础。

2.1 单例、并集与交集类型

首先来看一个例子：Erlang 标准库中的函数 `last_day_of_the_month/2`，它用于确定一个月有多少天。[1]

```
每月最后一天(_, 4) -> 30; 每月最后一天(_,
6) -> 30; 每月最后一天(_, 9) -> 30; 每月最
后一天(_,11) -> 30; 每月最后一天(Y, 2) ->
case is_leap_year(Y) of true -> 29; _ -> 28
end; 每月最后一天(_,M) -> 31.
```

在 Erlang 中，一个函数至少由一个函数子句组成。每个子句包含函数名和括号内的参数，后接箭头符号 ->以及函数体。子句之间用分号分隔，最后一个子句以句点结束。每个函数参数都是一个模式；在上面的例子中，这些模式包括常量（4、6.....）、通配符（_）和变量（Y）。常量只匹配其确切值，而通配符和变量则匹配所有内容。

Erlang 中的类型规范是可选的。可以使用 Erlang 类型规范语言为任何函数添加**-spec** 注解 [21]。Erlang 标准库为该函数提供了以下类型规范：[2]

```
-spec last_day_of_the_month(非负整数(), 1..12) -> 28 | 29 | 30 | 31
```

这里，类型 `1..12` 表示从 1 到 12 的整数范围。

虽然这个类型规范是正确的，但我们可以为该函数提供更精确的类型规范。为了细化 `last_day_of_month` 的类型，我们注意到，第一个参数（年份）可以是任何大于或等于零的整数，而第二个参数（月份）是介于 1 和 12 之间的整数，其中集合 {4,6, 9, 11} 中的值映射为 30，值 2 映射为 29 或 28，而 $\{1,\ldots,12\}\setminus\{4,6,9,11,2\} = \{1,3,5,7,8,10,12\}$ 中的值映射为 31。

上述函数最精确的类型——不允许在类型层级上进行算术运算——如图1所示。

该类型规范为`last_day_of_month`指定了三种函数类型，这意味着该函数实际上是一个接受三种不同类型参数的函数。符号; 在 Erlang 的类型语言中表示交集 。示例中使用的类型语言的其他特性还包括单例类型，例如2；表示数值集合 {0，1，2，...} 的类型非负整数()；两个类型的并集 $t_1 \mid t_2$；以及函数类型（t_1，...，t_n）-> t。Erlang 允许交集

[1] https://www.erlang.org/doc/man/calendar.html#type-ldom [2] 标准库实际上引入了一些类型同义词，我们在此直接将其展开。

```
-spec last_day_of_the_month (非负整数(), 2) -> 28 | 29; (非负整数(),
  4 | 6 | 9 | 11) -> 30; (非负整数(), 1 | 3 | 5 | 7 | 8 | 10 | 12)
-> 31.
```

图1. last_day_of_the_month 的精确类型规范。

```
-spec inter(整数()) ->整数(); ({原子(), 原子() | 整数()} |{整数(), 原子()}) -> 原子().
inter(X) ->case X of _ when is_integer(X) ->X + 1; {Y, Z} when is_atom(Y) ->Y; {_,
Z} ->Z end.
```

图2. inter 函数通过守卫和模式展示出现类型。

类型仅出现在类型规范的顶层，而不会嵌套在类型内部。并集的优先级高于函数箭头，而函数箭头的优先级又高于交集。所有这些类型运算符都是右结合的。

我们的静态类型检查器 Etylizer 支持这两种类型规范，并且都能成功检查该函数。因此，程序员可以在可读性和精确性之间权衡，而不损害类型安全。

2.2 类型测试、出现类型与穷尽性

Erlang 程序中一种常见的惯用写法是使用动态类型测试，根据值的运行时类型决定执行路径。与上文函数 `last_day_of_month` 中使用的函数子句守卫类似，我们可以在 **case** 子句中使用守卫，根据某个表达式的类型执行不同的分支。守卫是有效 Erlang 表达式的一个受限子集，其求值必须没有副作用。为了准确地对这类代码进行类型检查，类型系统必须能够通过模式匹配和类型测试（守卫），在不同分支中细化被匹配表达式的类型。

图 2 展示了一个示例，其中函数具有一个类型规范，该规范是两个不同函数子句的交集。

该规范确立了一种互斥的输入输出关系：整数输入会产生整数输出，而符合所指定元组变体任一形式的输入会产生原子输出。

出现类型用于验证 **case** 表达式的每个分支都符合类型签名。在第一个分支中，守卫 `is_integer(X)` 将 `X` 的类型从完整的输入并集细化为`integer()`，从而使操作 `X + 1` 类型安全。在第二个分支中，模式 `{Y, Z}` 与守卫`is_atom(Y)` 结合，选出了输入类型中 `X` 为元组且其第一个元素是原子的那一部分，即`{atom(), atom() | integer()}`。因此，返回值 `Y` 可确定为 `atom()`类型。

最后，只有未被前一个子句匹配的元组才会进入第三个分支。此时，`X` 的类型会细化为`{integer(), atom()}`，这是输入类型中剩余的部分。因此，可以确定变量 `Z` 的类型为 `atom()`，从而使返回类型

正确。此外，类型系统可以验证**case** 表达式是穷尽的，因为这三个分支共同涵盖了为输入**x**指定的所有可能类型。

在 Etylizer 代码库中，我们利用这些能力处理 Erlang 抽象语法树，其中包含大量分类讨论，最多可达 25 个子句。我们可以保证所有可能的类型表示都得到处理，而无需依赖兜底子句。据我们所知，目前没有其他系统支持 Erlang 的穷尽性检查，同时还能将守卫与出现类型、类型重建以及完整的集合论运算范围（并集、交集和否定）相结合。

2.3 递归数据类型与多态

Erlang 通过类型变量支持自定义类型定义和多态函数。请看标准库 `flatten` 函数，它来自 `lists` 模块：

```
flatten([H|T], 尾部) when is_list(H) ->flatten(H, flatten(T, Tail)); flatten([H|T], 尾部)
-> [H|flatten(T, Tail)]; flatten([], 尾部) -> 尾部.
```

Erlang 函数 `flatten` 会递归地展平嵌套列表，将所有元素合并为一个扁平列表，并追加给定的 尾部。其类型规范如下：

```
-spec flatten(DeepList, Tail) -> [term()] when DeepList :: [term() |
DeepList], Tail :: [term()].
```

“**when** ”子句引入类型约束，用于限制签名中类型变量允许的类型。在本例中，它指定`DeepList` 是深层列表，这是一种递归类型，其中每个列表元素要么是项（`term()`），要么是深层列表（`DeepList`）。请注意，在类型上下文中，竖线 `|` 表示类型并集，这与表达式上下文中使用的列表构造器`[H | T]` 不同。`Tail` 被约束为由项组成的普通列表。约束中的递归定义 `[term() | DeepList]`试图描述一种可以任意嵌套的结构。

此规格有三个显著局限。首先，尽管第一个参数的输入类型以递归方式指定，但该类型会简化为 `[term() | DeepList] = [term()]`。其次，该类型无法保留输入输出关系。输出类型 `[term()]` 范围过宽，也没有体现结果所包含的元素类型与输入类型之间存在某种关联。在对函数的应用进行类型检查时，这会造成问题。可以通过使用类型变量实现多态来解决此问题，从而表达输入类型与输出类型之间的关系。为表示这种关系，我们可以定义一个自定义递归类型：

```
-类型 deepList(A) :: [A | deepList(A)]. -规格
flatten(deepList(A), [A]) -> [A].
```

类型定义为类型表达式引入命名别名，从而提高可读性并支持重用。第三，该类型规范仍未能表达这样一项语义约束：展平结果中的元素本身不是列表。请看第一个函数子句中的调用`do_flatten(H, do_flatten(T, Tail))`。这里，列表构造器`[H|T]` 将列表分解为头部和尾部，而守卫`is_list(H)` 则检查头部本身是否为列表。`H` 的类型被细化为列表类型，但`do_flatten` 要求其第一个参数为`deepList`。使用这种通用类型规范时，实现无法通过类型检查。

```
-type deep_list(A) :: [etylizer:without(A, 列表类型) | deep_list(A)].
```

图 3. 使用集合论否定描述 flatten 的更精确递归类型。

要对 `flatten`进行类型检查，我们需要让规范更加精确。目前，Erlang 的类型语言还无法表达集合差。因此，我们需要使用自定义的集合论类型否定来表达该类型，如图 3所示。

使用这个经过细化的类型，函数就能通过类型检查。值得注意的是，此前所有针对 Erlang 的类型系统研究都未能成功验证 Erlang 标准库的 `flatten/2` 函数。

举一个结合递归、多态和交类型的例子：考虑 Erlang 标准库中的`filtermap/2` 函数，其规范如下：

```
-spec filtermap( fun((Elem) -> 布尔值() | {'true ', Value}), [Elem ] ) ->[Elem | Value]
when Elem :: term(), Value :: term(). filtermap(F, [Hd |Tail ]) ->case F(Hd) of true -> [Hd |
filtermap(F, Tail)]; {true,Val} -> [Val |filtermap(F, Tail)]; false ->filtermap(F, Tail) end;
filtermap(F, []) when is_function(F, 1) -> [].
```

顾名思义，`filtermap/2` 要么根据布尔函数筛选元素，要么在筛选的同时对筛选出的值应用额外的转换（从`T` 映射到`U`，或生成一个混合了转换后值与未筛选值的列表）。我们的类型检查器能够高效地判定这个高阶、多态递归函数的类型安全性。我们认为，它的实现可以用更精确的类型来表达：

```
-spec filtermap(fun((T) -> 布尔值()), [T]) -> [T]; (fun((T) -> {true , U} | false), [
T]) -> [U]; (fun((T) -> {true , U} | 布尔值()), [T]) -> [T | U].
```

我们的系统也成功验证了这个类型规范。

不过，某些高级多态数据类型并不受支持。请看下面这个嵌套数据类型的例子 *nesteddatatype* [5, 26]：

```
-type perfect(A) :: A | perfect({A, A}).
```

本质上，`perfect(A)` 描述的是叶节点类型为`A`的完全二叉树。这个例子有趣之处在于，在等式右侧，`perfect({A,A})`的参数化方式与定义左侧不同。我们的类型系统无法处理类型上的这种多态递归。

此外，我们的系统不支持二阶多态，而要为以多态函数作为参数的函数进行类型推断，就必须支持二阶多态。正如第 4.3 节所述，这一限制会影响某些高阶编程模式，尽管这些模式在实践中并不常见。

2.4 局部函数的类型重构

在 Erlang 中，一次性使用的辅助函数通常定义为匿名函数。然而，由于 Erlang 不支持为匿名函数添加类型注解，因此必须将这些函数重构为单独的具名函数，才能进行类型检查，这会使类型检查变得更加复杂。一个显著的例子是对

```
-spec dnf_intersect(dnf(), dnf()) -> dnf(). -spec intersect( {dnf(), #{integer() =>
dnf()}}, {dnf(), #{integer() => dnf()}} ) -> {dnf(), #{integer() => dnf()}}.
intersect({DefaultT1 , T1}, {DefaultT2 , T2}) -> AllKeys = maps:keys(T1) ++
maps:keys(T2), IntersectKey = fun(Arity) -> dnf_intersect( maps:get(Arity , T1,
DefaultT1), maps:get(Arity , T2, DefaultT2) ) end , L = lists:map(fun(Key) -> {Key,
IntersectKey(Key)} end, AllKeys), { dnf_intersect(DefaultT1 , DefaultT2),
maps:from_list(L) }.
```

图 4. Etylizer 代码库中以析取范式表示的多元数函数类型的交集，以及为匿名函数 IntersectKey 重建的类型。

Etylizer 实现中多元数函数类型的内部表示；图 4 展示了以析取范式表示的集合论类型的交集运算。

该表示由一个元组构成，其中包含所有未指定元数的默认表示，以及从元数到函数表示的显式映射。函数`dnf_intersect` 实现了两个函数空间的交集运算，并能正确处理显式定义的元数和默认情况。在此函数中，`IntersectKey` 是一个没有类型注释的匿名函数，用于计算多元数函数中给定元数对应的交集。Etylizer 成功重建了它的类型，并验证了该函数的类型安全性。

2.5 值依赖类型

Erlang 类型检查的核心难点在于，类型有时取决于值。标准库规范凸显了这一挑战：

```
-spec element(N :: pos_integer(), tuple()) -> 项(). -spec ukeysort(N :: pos_integer(), [元组]) -> [元组]
when 元组 :: 元组().
```

这两个函数会动态访问元组元素：`element/2` 用于单个元组，`ukeysort/2` 则根据动态索引对元组列表排序。请看它们在 Erlang 标准库中的`from_list/1` 以及 Etylizer 代码库中的`loc_exp/1` 的用法：

```
-spec from_list([{Key , Value}]) -> 有序字典(Key , Value). from_list([]) -> []
from_list([{_,_}] = 对) -> 对; from_list(对) -> lists:ukeysort(1, reverse_pairs(对 ,
[])).


-spec loc_exp(exp()) ->loc(). loc_ex
p(X) -> 元素(2, X).
```

函数 `from_list/1` 依赖 `lists:ukeysort/2` ，按特定元组位置对元组列表进行排序；而 `loc_exp/1` 则从大小未知的元组中提取第二个元素。两者都需要知道特定索引，才能确定精确的类型。

在我们的实现中，我们通过 类型覆盖来处理这些情况，为常见模式提供专门的规范：

```
% 针对常量索引的 lists:ukeysort 覆盖-spec 'lists:ukeysort'(
1, [{K, V}]) -> [{K, V}].

% 针对常量索引的 element 覆盖-spec 'erlang:element'
(2, {_A, B}) -> B; (2, {_A, B, _C}) -> B; (2, {_A, B, _C,
_D}) -> B; ...; (_, tuple()) ->term().
```

这些覆盖实际上表达了类型层级上的模式匹配，可验证输入是否具有足够的大小，以及是否具备给定索引所要求的正确结构。我们的实证研究（第D节）表明，尽管 Erlang 中依赖值的操作具有动态特性，但大多数用法（76%）使用常量参数，因此能够利用我们系统的特性进行静态类型检查。虽然任何可能的`element/2` 和 `ukeysort/2`应用都需要通用的依赖值类型，但我们结合使用单例类型与交类型，足以应对实际中常见的情况。

这些示例表明，集合论类型与 Erlang 的类型语言天然契合。我们的经验表明，尽管某些现有类型签名可能需要完善，但 Erlang 的所有语言构造都可以得到支持，从而准确反映该语言的语义。尤其是，在 Erlang 中常用于函数子句和 **case** 表达式的类型检查和守卫表达式，都可以进行静态验证，避免通常会导致语义错误的常见失误。

3 形式化类型系统

本节为我们针对 Erlang 的类型系统提供形式化基础。我们定义了语言 λ_{erl}，这是一种核心演算，旨在捕捉 Erlang 顺序编程的关键要素。它在 λ-演算的基础上扩展了常量、对、带类型测试的模式匹配，以及顶层相互递归函数。 λ_{erl} 的类型系统基于子类型关系、多态集合论类型 [14, 23], 和出现类型 [45]。其表述受到 Castagna 及其同事将集合论类型用于建模多态变体 [12]的研究启发。

3.1 语法

图 5 定义了 λ_{erl}的语法。我们使用 α， $\beta \in \mathbb{V}$ 表示类型变量； x， $y \in \mathbb{X}$ 表示表达式变量； $\kappa \in \mathbb{K}$ 表示常量； $\iota \in \mathbb{S}$ 表示单例类型。我们假设集合 $\mathbb{V}$， $\mathbb{X}$， $\mathbb{K}$和 $\mathbb{S}$ 均为可数无限集。常量集合 $\mathbb{K}$包含整数和浮点数，而单例类型集合 $\mathbb{S} \subseteq \mathbb{K}$ 仅包含整数。元变量 A 的取值范围为有限类型变量集。

记号 $\mathfrak{s}_{i \in I}$ 表示某个语法构造 $\mathfrak{s}$和某个指标集 $I = \{1、\ldots\ldots、n\}$ 所对应的序列 $\mathfrak{s}_1 \ldots\ldots \mathfrak{s}_n$。

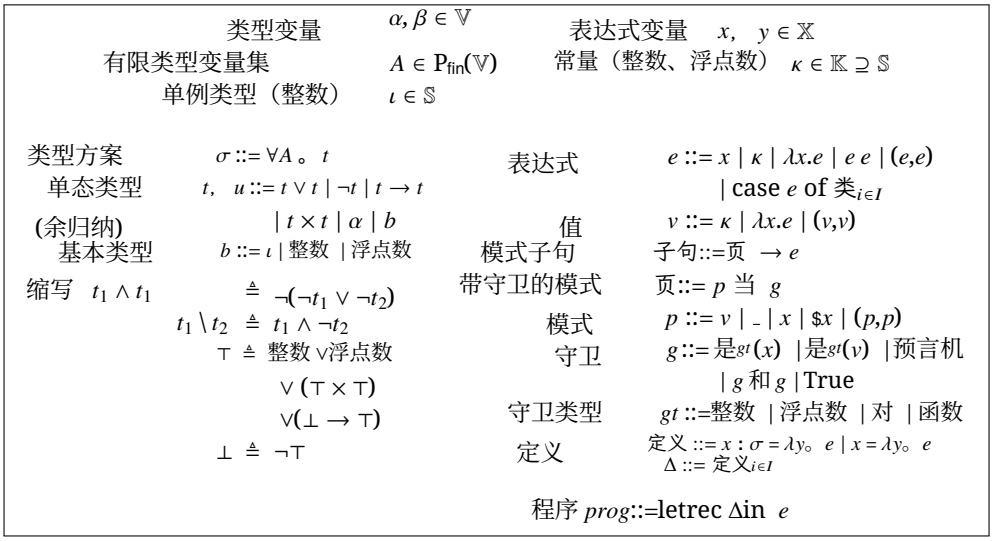

图 5. λ_{erl}的语法

3.1.1 类型

类型方案 $\sigma = \forall A \。\ t$ 表示对有限个类型变量A上的单态类型 t 进行量化。在某些地方，我们将类型方案 $\forall \emptyset \。\ t$ 与单态类型 t视为相同。

单态类型 t 涵盖集合论连接词：并集 $t_1 \vee t_2$ 和否定 $\neg t$（稍后将予以解释），以及用于函数 $t_1 \to t_2$、对 $t_1 \times t_2$、类型变量 α和基本类型 b的类型构造子。基本类型包括表示单个整数的单例类型 ι，以及表示所有整数和浮点数的类型。

集合论类型需要作一些进一步说明。这里我们只勾勒主要思想，详情请参阅相关文献 [10–12, 14, 23]。在集合论类型中，我们将类型解释为某个模型的子集$\mathcal{M}$。于是，类型之间的子类型关系就成为集合包含。直观而言，$\mathcal{M}$ 可以是所研究编程语言中所有类型良好的值构成的集合；类型 t 则应表示该类型的所有值构成的集合。出于技术上的原因，这样定义 $\mathcal{M}$ 在实践中效果不佳。幸运的是，还有其他定义方式，使我们能够得到直观的模型 [8, 23]。

一般而言，集合论类型的模型定义与特定编程语言紧密相关，尤其取决于其求值方式（严格或非严格）以及可用的类型构造器。不过，Erlang 与上述文献所研究的编程语言足够相近，因此，[10–12] 的工作中提出的模型和算法可以直接应用于我们的场景。

回到单态类型的语法，$t_1 \vee t_2$ 表示与 t_1 和 t_2关联的集合的并集，而 $\neg t$ 表示相对于整个模型而言 t 的补集。交集 $t_1 \wedge t_2$、集合差 $t_1 \setminus t_2$，以及顶类型 $\top$ 和底类型$\bot$ 都定义为缩写，见图 5。顶类型 $\top$ 表示整个模型；也就是说，它是整数、浮点数、对的顶类型 $\top \times \top$以及函数的顶类型$\bot \to \top$的并集。底类型 $\bot$表示空集，定义为 $\top$的补集。按照约定，类型连接符 $\vee$、 $\wedge$、 $\neg$ 以及类型构造子 $\to$、 $\times$均向右结合，而（由高到低的）优先级顺序为 $\vee$、 $\wedge$、 $\neg$、 $\to$、 $\times$。

为了支持递归类型，单态类型 t 的定义应按余归纳方式理解。因此，单态类型是一棵可能无限的树。例如，列表类型可以定义为满足以下方程的类型：列表 $\alpha = 0 \vee (\alpha \times \text{列表}\ \alpha)$。这里，单元素集合 0 表示空列表。记法“列表 α =”并非类型语法的一部分，但它使我们能够用有限的描述来表示无限树 $0 \vee \alpha \times (0 \vee \alpha \times .. .)$。

我们要求这些可能无限的树具有正则性和收缩性 [12, 37]。正则性要求一棵树只包含有限个不同的子树。这一条件对于确立子类型算法的可判定性至关重要。收缩性意味着每个无限分支上，类型构造子 $\rightarrow$ 和 $\times$都出现无限多次。这一条件排除了满足诸如 $t = t \vee t$ 或 $t = \neg t$这样的方程的荒谬类型。

Erlang 的类型语法仅以有限方式支持交类型；也就是说，交类型只能作为顶层函数类型说明中的最外层连接词。此外，Erlang 不提供否定类型的语法。不过，我们的形式化支持完整的集合论类型，因为若干类型规则（见 3.3）需要其表达能力。

3.1.2 表达式、定义和程序

表达式 e 包括变量 x、整数和浮点数的常量 κ、λ 抽象 $\lambda x.\, e$、函数应用 $e_1\, e_2$、对 (e_1,e_2)，以及用于模式匹配的 case 表达式。按照惯例，函数应用向左结合，而 λ 抽象的主体尽可能向右延伸。值 v 包括常量、λ 抽象和值对。

表达式 case e of $cls_{i \in I}$ 将审查表达式 e 与模式子句 cls_i进行匹配。模式子句的形式为 p when $g \rightarrow e'$，其中 p 是模式，g 是守卫，e' 是主体表达式。模式由值 v、通配符、绑定变量 x、已绑定变量 $\$x$以及对模式（$p$,$p$）组成。模式中的值和变量还需要进一步说明。

在 Erlang 中，模式变量遵循一种颇为特殊的作用域规则。如果变量已在外层作用域中绑定，它就相当于常量模式，只匹配与该变量的值相等的值。如果变量未在外层作用域中绑定，它总是匹配成功，并将待匹配的值绑定给该变量。

在我们的演算中，我们通过显式构造处理这两种情况。已绑定模式变量 $\$x$ 要求 x 处于作用域内；与 $\$x$ 匹配，就是与 x 所绑定的值进行匹配。绑定模式变量 x 总是匹配成功，并将该值绑定到 x，即使 x 已在外层作用域中定义也是如此。随后，（内部的） x 的作用域涵盖相应模式子句的守卫和主体。

已绑定模式变量所蕴含的语义促使我们将值（而不仅仅是常量）纳入模式语法。在求值过程中，已绑定模式变量$\$x$应替换为某个值 v。

守卫 g 进一步约束匹配。它本质上是针对守卫类型 gt 的 True、预言机和类型测试的合取。类型测试写作 $\text{is}_{gt}(x)$ 或 $\text{is}_{gt}(v)$，对应于 Erlang 中的守卫函数，例如 `is_integer`。[3] 与 Erlang 类似，守卫类型仅限于整数、浮点数、对和函数。守卫还包括预言机，用于模拟无法静态分析的守卫表达式

[3]我们只支持在守卫测试中使用变量和值，因为对真实世界 Erlang 代码的实证评估显示，几乎所有类型测试都是针对变量进行的（第 C 节）。其他类型的测试需要建模为预言机。

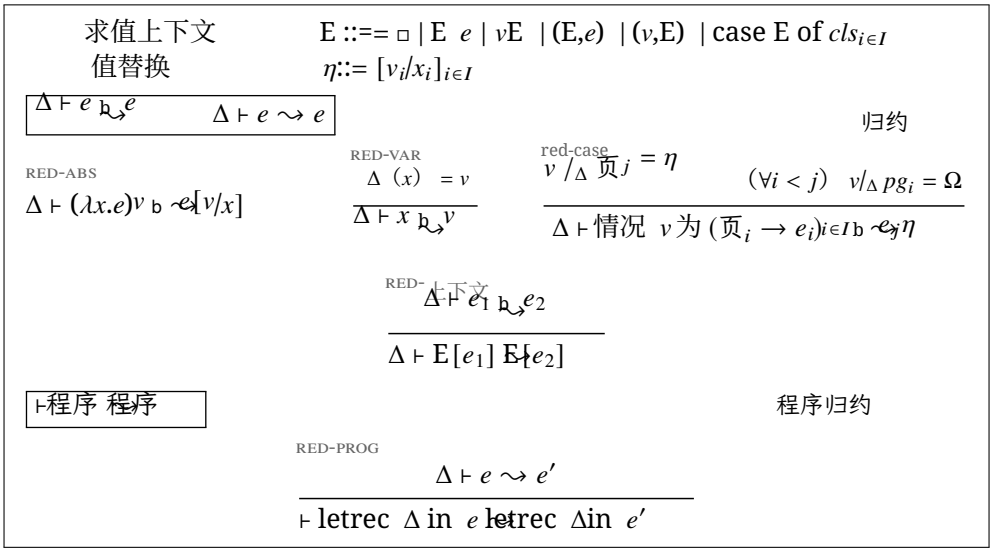

图 6. λ_{erl}的动态语义

（例如 Erlang 中的 tuple_size）。在某些地方，我们将模式 p 记作带守卫模式 p，其中守卫条件为 True。

顶层定义 定义 将变量绑定到函数，并可选择附带类型注释 σ。定义序列记作 Δ，并假定 Δ 中同一变量没有重复定义。我们用 定义域(Δ) 表示在 Δ 中定义的变量集合，并用$\Delta(x)$ 访问 Δ 中 x 的定义的右侧，假定 $x \in$ 定义域(Δ)。因此，程序程序 由相互递归的函数定义序列 Δ 和主表达式 e 组成。

3.1.3 其他事项

我们记 ftv(t) 表示类型 t中自由类型变量的集合。关于 ftv(t) 的余归纳定义见第 A节，定义A.1。我们记 fv(e) 和 fv(pg) 分别表示表达式 e 和带守卫模式 pg中自由表达式变量的集合。这些定义也见第 A节，定义A.5。我们将所有语法构造视为在对绑定的表达式变量和类型变量重命名后相等。

3.2 动态语义

动态语义的形式化分布在两幅图中。图6 定义了归约关系，图7 定义了用于对模式匹配求值的辅助定义。

图6 为 λ_{erl}定义了标准的按值调用小步归约关系。按值调用求值上下文$\mathcal{E}$ 是一个带有空位 □的表达式，该空位标记了下一求值步骤应发生的位置。我们用$\mathcal{E}[e]$ 表示将$\mathcal{E}$ 中的空位替换为表达式 e。

元变量 $\eta = [v_i/x_i]_{i \in I}$ 表示将表达式变量 x_i 替换为值 v_i 的避免捕获的替换。我们用 $e\eta$ 表示将替换 η 应用于表达式 e（其他语法构造也类似），用 定义域(η) 表示 η 的定义域，并用 $\eta(x)$ 表示访问变量 x 对应的值，这里隐含假设 $x \in$ 定义域(η)。

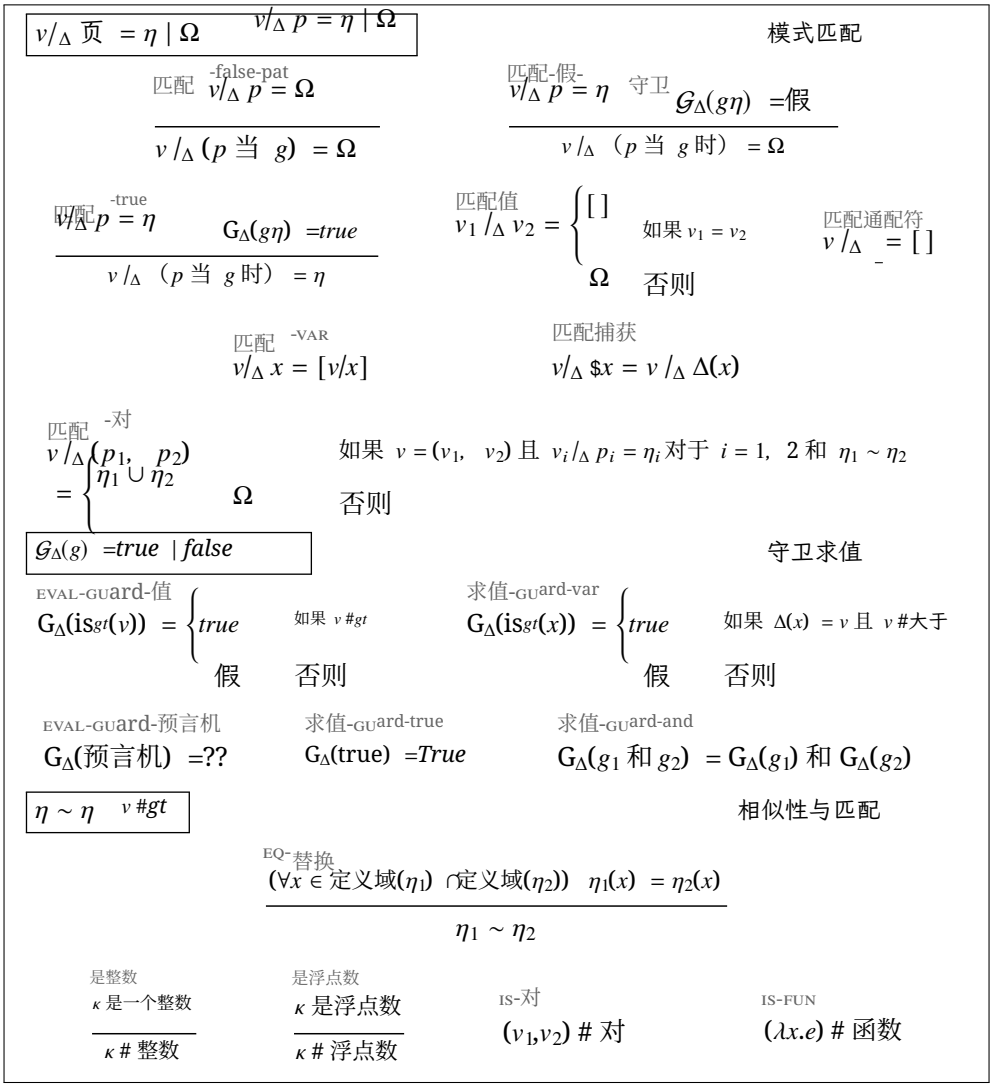

图 7. 模式的动态语义

关系 $\Delta \vdash e \to_\beta e'$ 是基本求值关系，它在 e 的顶层执行从 e 到 e' 的单个归约步骤。规则 red-abs 按照通常方式归约 λ 表达式。规则 red-var 要求 Δ 访问 letrec 绑定变量，因为对 letrec 绑定变量的替换是惰性执行的，以支持相互递归函数。规则 red-case 将 case 表达式归约为第一个匹配其带守卫模式的分支的主体。它使用 $v /_\Delta \mathrm{pg}$ 将值 v 与带守卫模式 pg 进行匹配，成功时产生替换 η，失败时产生 Ω。下面将解释 $v /_\Delta \mathrm{pg}$（另见图 7）。

关系 $\Delta \vdash e \leadsto_\beta e'$ 在求值上下文的引导下，从 e 归约一步到 e'，参见规则 RED-CONTEXT。关系 $\vdash prog \; prog'$ 通过归约程序的主表达式，使程序 $prog$ 经过一步归约到 $prog'$，参见规则 RED-PROG。

图 7 定义了函数 $v/_{\Delta} pg$ 和 $v/_{\Delta} p$，用于刻画模式匹配的语义。如果值 v 匹配带守卫模式 pg 或模式 p，那么结果就是对模式绑定变量的代换 η。否则，结果为错误 Ω。

前三条规则将带守卫模式 p when g 与一个值进行匹配。这三条规则都会先匹配模式。如果匹配成功，规则 MATCH-FALSE-GUARD 和 MATCH-TRUE 会使用 $\mathsf{G}_{\Delta}(g\eta)$ 检查守卫求值结果是否为*True*。

规则 match-val 将值 v_1 与模式值 v_2 进行匹配。只有当 v_1 和 v_2 相等时（忽略绑定变量的重命名），匹配才会成功。规则 match-wild 和 match-var 分别用于匹配通配符和变量。规则 match-capture 处理已绑定模式变量 $\$x$。由于求值过程中会替换 λ 绑定的变量，这类变量必须在顶层绑定。因此，该规则会获取 Δ 中 x 的定义，记作 $\Delta(x)$。规则 match-pair 处理对模式（p_1,p_2）。如果 p_1 和 p_2 都绑定同一变量，那么被匹配的值必须相等，这表示为 $\eta_1 \sim \eta_2$。记号 $\eta_1 \cup \eta_2$ 表示 η_1 和 η_2 的并集，当 $\eta_1 \sim \eta_2$ 时该并集定义良好。代换的相似性 $\sim$ 定义在图 7 的下半部分。

对模式守卫 g 求值的结果记作 $\mathsf{G}_{\Delta}(g)$。规则 EVAL-GUARD-VALUE 通过检查值 v 是否属于某个守卫类型 gt，来对类型测试 $\mathsf{is}_{gt}(v)$ 求值，记作 $v \# gt$（见图 7底部）。规则 EVAL-GUARD-VAR处理对变量进行的类型测试。规则 EVAL-GUARD-ORACLE将预言机的结果留作未指定，记作 ??，以强调类型系统不得依赖其结果这一事实。[4]

3.3 静态语义

本节将为 λ_{erl}形式化一个声明式类型系统。我们依赖于集合论类型之间的子类型关系，记作 $t_1 \leq t_2$。该关系由 [14] 和 [10]定义。两个单态类型 t 和 t' 等价，记作 $t \simeq t'$，当且仅当 $t \leq t'$ 和 $t' \leq t$成立。

类型环境 $\Gamma = \{x_i : t_i \mid i \in I\}$ 是从表达式变量到（单态）类型的偏映射。我们用 $\emptyset$ 表示空类型环境，用 $\mathsf{dom}(\Gamma)$ 表示Γ的定义域，并用 $\Gamma(x)$ 表示在 Γ中查找 x的类型，默认假设 $x \in \mathsf{dom}(\Gamma)$。变量环境的扩展记作 $\Gamma, x : t$，其中新的绑定 $x : t$ 会遮蔽 Γ中可能存在的 x 绑定。类似地，在串接 Γ, Γ' 中，Γ' 中的绑定优先于Γ中的绑定。记号 $\Gamma \neq \bot$要求不存在满足 $\Gamma(x) = \bot$ 的 $x \in \mathsf{dom}(\Gamma)$。

与类型环境类似，我们将方案环境 $\Sigma = \{x_i : \sigma_i \mid i \in I\}$ 定义为从表达式变量到类型方案的偏映射；其他定义和记法均与此类似。此外，我们假设类型方案 $\forall A.\, t$ 仅包含有意义的类型变量；也就是说，对于所有类型变量 $\alpha \in A$，都有 $t[\bot/\alpha] \neq t$。关于有意义类型变量的形式化处理，请参见 [12]。

3.3.1 类型守卫模式

图 8 给出了计算带守卫模式所得类型环境的定义：$t /\!\!/ pg$ 和 $t /\!\!/ p$ 分别针对类型为 t 的值进行匹配时，计算绑定在pg 中的变量以及 p中变量的环境 Γ；而 $\mathsf{env}(g)$ 则计算由守卫 g产生的环境。除三种情况外，这些定义都很直接

[4]虽然 Erlang 完整的守卫系统包括特定元数函数检查（例如 is fun/2），但我们的形式化中省略了这部分复杂性，因为它不会带来理论上的新见解。实际上，为我们的实现扩展支持元数感知的函数守卫非常简单。

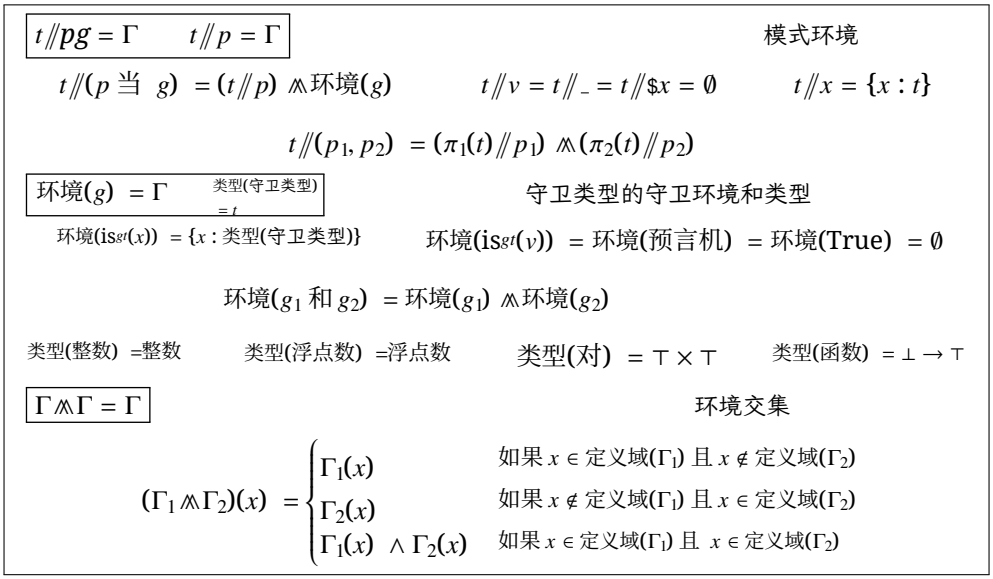

图 8. 带守卫的模式所产生的环境

。首先，如果两个环境 Γ_1 和 Γ_2 都包含同一变量的类型，我们就对这两个类型取交集。环境交集记作 $\Gamma_1 \curlywedge \Gamma_2$，定义见图 8 底部。其次，ty(gt) 给出与守卫类型 gt 对应的类型 t。在其定义中，$\top \times \top$ 和 $\bot \to \top$ 分别是所有对类型和函数类型的超类型。第三，$t /\!\!/ (p_1, p_2)$ 的定义使用 $\pi_1(t)$ 和 $\pi_2(t)$，将对类型 $t \leq \top \times \top$ 的左、右分量分别投影出来（参见 A 节、性质 A.2）。

图9 为带守卫模式 pg定义了两种类型：

- $[\![pg]\!]^{\uparrow}$ 是潜在类型：如果某个表达式匹配pg，那么该表达式必定属于此类型。
- $[\![pg]\!]^{\downarrow}$ 是接受类型：如果某个表达式属于此类型，那么pg 一定匹配该表达式。

在解释这两种类型之前，我们还需要一些辅助定义。首先，如图 9 底部的定义所示，pbv(p) 表示由模式 p 绑定的表达式变量集合，而ty(κ) 表示某个常量 κ 的类型。

其次，$[\![p]\!]^{\uparrow}_{\Gamma}$ 和 $[\![p]\!]^{\downarrow}_{\Gamma}$ 分别表示相对于类型环境 Γ 的某个模式 p 的潜在类型和接受类型。实际应用中，Γ 是由该模式的守卫产生的类型环境。

某个模式 p 的潜在类型与带守卫的模式的直觉相同：如果一个表达式匹配 p，那么匹配的表达式就具有类型 $[\![p]\!]^{\uparrow}_{\Gamma}$。因此，如果 p 是一个（数值）常量 κ，那么匹配的表达式必须具有 ty(κ) 类型。如果 p 是一个函数值或捕获变量，我们就用顶类型 $\top$来近似其潜在类型。请注意，实践中对函数进行模式匹配并没有太大用处，但为了完备性，我们将其纳入其中。

对于接受类型，如果某个表达式的类型为 $[\![p]\!]^{\downarrow}_{\Gamma}$，那么 p 肯定与该表达式匹配。因此，如果 p 是整数常量 κ，那么接受类型就是 κ。对于浮点常量，不存在单例类型，因此我们无法给出精确的接受类型。在这种情况下，以及对于函数和捕获变量，我们

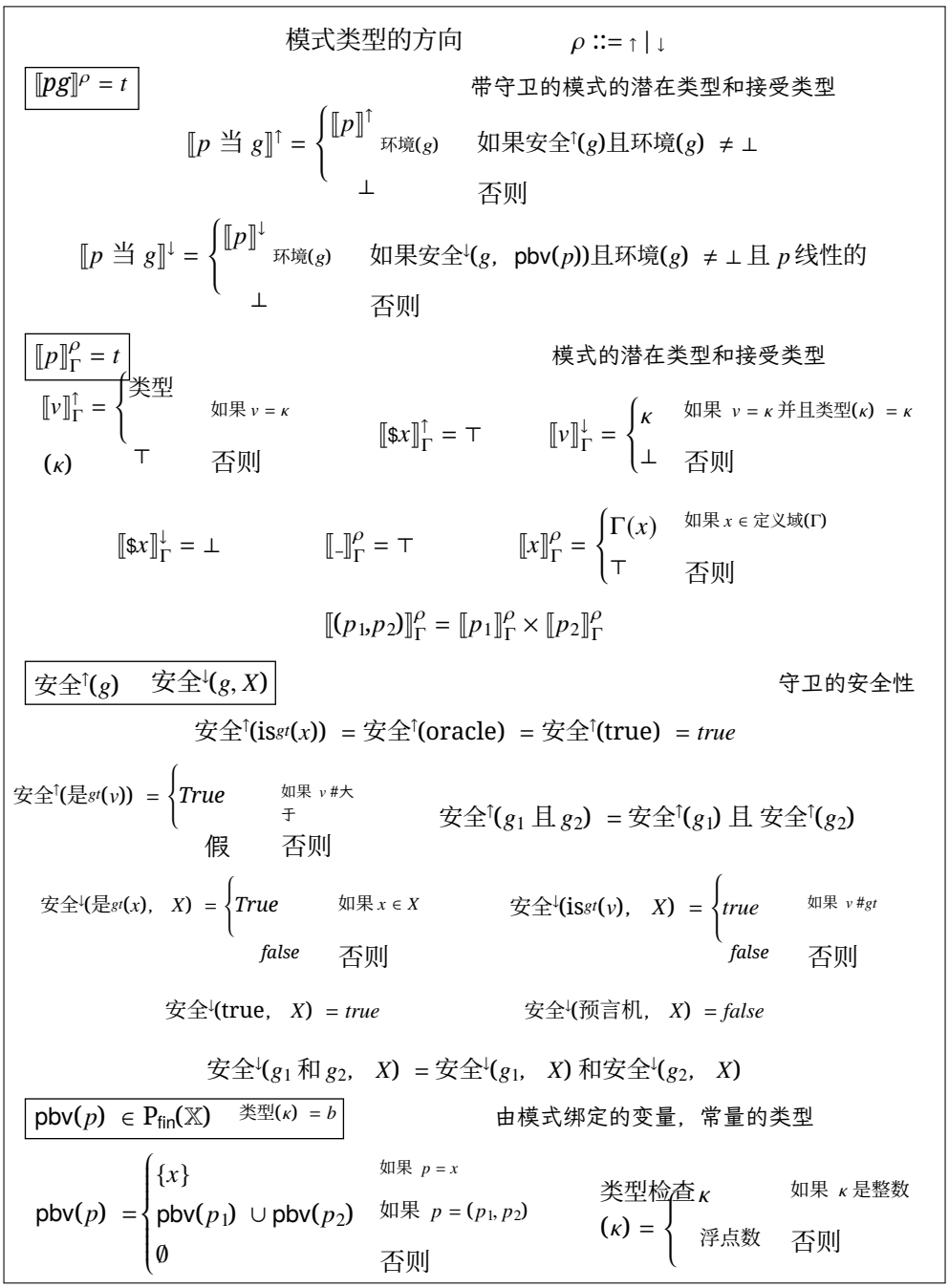

图 9. 类型守卫模式

保守地使用底类型 $\bot$ 作为接受类型。

通配符模式始终匹配，因此潜在类型和接受类型都是 $\top$。变量模式也会无条件匹配；如果该变量在 Γ 中绑定，我们就使用 Γ 中的类型作为潜在类型和接受类型，否则使用 $\top$。对于对类型，我们递归处理。

第三个也是最后一个辅助项是谓词安全，它对某个守卫的结果提供静态近似。

- 安全$^{\uparrow}(g)$ 是假当 g 永远不会成功；否则为真。本质上，结果为假仅当 g 包含一个类型测试是$_{gt}(v)$ 使得 $v \# gt$ 不成立时才有可能。（# 的定义见图7。）

- 安全$^{\downarrow}(g, X)$ 当 g 始终能够成功时为 真，否则为假。这里，X 是模式 pg 中模式 p 所绑定的变量集合。其定义中值得关注的情况是，g 的形式为 $\mathsf{is}_{gt}(x)$。如果 $x \in X$，那么结果为真，因为模式 g 所属的模式会绑定 x，而该模式的接受类型会限制 x 的类型，使守卫能够成功。否则，$x \notin X$，因此我们需要采取保守处理，结果为假。

最后，我们回到某个带守卫模式$pg = p$ when g 的潜在类型和接受类型（见图9顶部）。潜在类型 $[\![pg]\!]^{\uparrow}$ 是相对于环境$\mathsf{env}(g)$而言，p 的潜在类型，前提是守卫有可能匹配。否则，潜在类型为 $\bot$。要使守卫 g 有可能匹配，我们要求$\mathsf{safe}^{\uparrow}(g)$且 $\mathsf{env}(g) \neq \bot$。后一条件确保环境$\mathsf{env}(g)$将所有变量绑定到非空类型；否则，$g$会对某个变量 x包含冲突的类型测试，从而导致$\mathsf{env}(g)(x) = \bot$。

接受类型 $[\![pg]\!]^{\downarrow}$ 是相对于环境$\mathsf{env}(g)$而言，p 的接受类型，前提是对于 p绑定的变量集合，守卫总能成功。为此，我们要求$\mathsf{safe}^{\downarrow}(g, \mathsf{pbv}(p))$且$\mathsf{env}(g) \neq \bot$，并且 p 必须是线性的。（线性模式不包含对同一变量的重复绑定。）这里有两点值得注意。首先，守卫 g被标记为安全，即使它包含针对 p绑定的变量的类型测试 $\mathsf{is}_{gt}(x)$。这是可以的，因为 $[\![p]\!]^{\downarrow}_{\mathsf{env}(g)}$ 随后会将 x 的类型限制为$\mathsf{ty}(gt)$。其次，要求 p 具有线性，是因为否则与同一变量匹配的值必须相等（见图7，规则MATCH-EQUAL）。然而，静态判断这种相等性超出了我们的类型系统的作用范围。

我们用三个例子来结束对带守卫模式类型规则的讲解。考虑下面的 case 表达式，其中 x 和 y 是类型未知的变量。

$$\texttt{case}\ (x, y)\ \texttt{of}\ (1, z)\ \texttt{when}\ \mathsf{is}_{\mathtt{int}}(z) \to \ldots$$

定义 $g_1 := \mathsf{is}_{\mathtt{int}}(z)$。守卫的环境为 $\mathsf{env}(g_1) = \{z : \text{整数}\}$。此外，我们有 $\mathsf{safe}^{\uparrow}(g_1) = \mathit{true} = \mathsf{safe}^{\downarrow}(g_1, \{z\})$。因此，潜在类型与接受类型相等：

$$[\![(1, z)\ \texttt{when}\ g_1]\!]^{\rho} = [\![(1, z)]\!]^{\rho}_{\{z:\mathtt{int}\}} = 1 \times \mathtt{int} \qquad (\text{for } \rho = \uparrow, \downarrow)$$

现在，我们将守卫改为对 y 进行类型测试：

$$\texttt{case}\ (x, y)\ \texttt{of}\ (1, z)\ \texttt{when}\ \mathsf{is}_{\mathtt{int}}(y) \to \ldots$$

定义 $g_2 := \mathsf{is}_{\mathtt{int}}(y)$。守卫的环境为环境$(g_2) = \{y : \text{整数}\}$。此外，$\text{安全}^{\uparrow}(g_2) = \mathit{true}$ 且$\text{安全}^{\downarrow}(g_2, \{z\}) = \mathit{false}$，因为存在一个针对未由模式绑定的变量进行的类型测试。因此，我们得到

$$[\![(1, z)\ \texttt{when}\ g_2]\!]^{\uparrow} = [\![(1, z)]\!]^{\uparrow}_{\{y:\mathtt{int}\}} = 1 \times \top$$
$$[\![(1, z)\ \texttt{when}\ g_2]\!]^{\downarrow} = \bot$$

我们的类型系统不会为此类情况提供更精确的类型，因为实证评估表明，在真实世界的 Erlang 程序中，类型测试几乎总是针对由模式绑定的变量进行的（见第 C节）。

对于最后一个示例，我们再次更改守卫。现在它包含 oracle，表示守卫中无法进行静态分析的部分。

$$\texttt{case}\ (x, y)\ \texttt{of}\ (1, z)\ \texttt{when}\ \texttt{is}_{\texttt{int}}(z)\ \texttt{and}\ \texttt{oracle} \to \ldots$$

定义 $g_3 := \text{是}_{\texttt{int}}(z)$ 和 oracle。环境为 环境(g_3) $= \{z : \text{整数}\}$，并且我们有安全$^{\uparrow}(g_3)$ $=$ 真 且 安全$^{\downarrow}(g_3,\{z\})$ $=$ 假。因此

$$[\![(1, z)\ \texttt{when}\ g_3]\!]^{\uparrow} = [\![(1, z)]\!]^{\uparrow}_{\{z:\texttt{int}\}} = 1 \times \texttt{int}$$
$$[\![(1, z)\ \texttt{when}\ g_3]\!]^{\downarrow} = \bot$$

所有与该模式匹配且通过守卫的表达式，其类型（潜在类型）必定为 $1 \times$ 整数。但由于 oracle 的存在，我们无法静态保证此类型的任何表达式实际上都能通过守卫（接受类型 $\bot$）。

3.3.2 表达式与程序的类型判定

图 10展示了 λ_{erl} 的类型规则。判断 $\Sigma{\cdot}\Gamma \vdash e : t$ 断言，在环境Σ和 Γ 下，表达式 e 的类型为 t。采用两个环境的原因是，Σ 包含顶层`letrec` 定义中的多态绑定，而 Γ 包含由 λ 或模式匹配绑定的变量的单态绑定。

常量 D-CONST、抽象 D-ABS、应用 D-APP、对 D-PAIR以及包含规则 D-SUB 的规则都是标准的。变量有两条规则。规则 D-VAR 在 Γ 中查找变量的类型，而规则 D-VAR-POLY 从 Σ 中获取类型模式并将其实例化。这两条规则要求变量包含在 Σ 或 Γ 中，但绝不能同时包含在两者中。通过重命名受约束变量，我们总能满足这一要求。类型模式的实例集合定义如下：

$$\mathsf{inst}(\forall A\,.\,t) = \{t\theta \mid \mathsf{dom}(\theta) = A\}$$

这里，$\theta = [t_i/\alpha_i]_{i\in I}$ 表示用类型 t_i 代换类型变量 α_i 的避免捕获的类型替换。我们用 $t\theta$ 表示将 θ 应用于某个类型 t，用 $\mathsf{dom}(\theta)$ 表示其定义域，并用 $\mathsf{ftv}(\theta)$ 表示 θ 的值域中自由类型变量的集合。

规则 D-CASE 为 case 表达式确定类型。第一个前提为审查表达式 e 赋予类型 t。第二个前提检查 case 的分支是否穷尽：t 必须是所有分支的接受类型的并集 $\vee_{i\in I}[\![pg_i]\!]^{\downarrow}$ 的子类型。

其余前提必须对所有分支都成立 $i \in I$。对 $\mathsf{fv}(pg_i)$ 的限制确保 pg_i 中使用的所有变量都已在某处定义。类型 t_i 是分支 i的输入类型 t_i；也就是说，如果第 i 个分支匹配，我们就可以假定审查表达式的类型。为计算 t_i，我们从 t（审查表达式的完整类型）中减去所有前序分支的接受类型 $\vee_{j<i}[\![pg_j]\!]^{\downarrow}$，因为在匹配第 i 个分支时，这些分支不可能已经匹配。然后，我们与该分支自身的潜在类型 $[\![pg_i]\!]^{\uparrow}$ 取交集，因为匹配意味着 t_i 是潜在类型的子类型。

Γ_i 是用于对 i个分支的主体进行类型检查的环境。它是 Γ与通过将 pg_i 与类型为 t_i 的值进行匹配而得到的环境的交集，记作 $\Gamma \curlywedge t_i /\!\!/ pg_i$。我们将这两个环境取交集，以便对 Γ 中已有变量的类型通过模式匹配进一步细化。关于 8 中 $\curlywedge$和 $t_i /\!\!/ pg_i$的定义，请参见图。

$\boxed{\Sigma \cdot \Gamma \vdash e : t}$ 表达式类型判定

D-VAR
$$\frac{\Gamma(x) = t \qquad x \notin \text{定义域}(\Sigma)}{\Sigma \cdot \Gamma \vdash x : t}$$

d-var-poly
$$\frac{t \in \text{inst}(\Sigma(x)) \qquad x \notin \text{定义域}(\Gamma)}{\Sigma \cdot \Gamma \vdash x : t}$$

D-常量
$$\Sigma \cdot \Gamma \vdash \kappa : \text{类型}(\kappa)$$

D-ABS
$$\frac{\Sigma \cdot \Gamma, \ x : t_1 \vdash e : t_2}{\Sigma \cdot \Gamma \vdash \lambda x.e : t_1 \to t_2}$$

D-APP
$$\frac{\Sigma \cdot \Gamma \vdash e_1 : t' \to t \qquad \Sigma \cdot \Gamma \vdash e_2 : t'}{\Sigma \cdot \Gamma \vdash e_1\, e_2 : t}$$

d-case
$$\frac{\begin{array}{l}\Sigma \cdot \Gamma \vdash e : t \qquad t \le \bigvee_{i \in I} [\![pg_i]\!]^{\downarrow} \\ (\forall i \in I) \quad \text{自由变量}(\text{程序}_i) \subseteq \text{定义域}(\Sigma) \cup \text{定义域}(\Gamma) \\ t_i = (t \setminus \bigvee_{j<i} [\![\text{页}_j]\!]^{\downarrow}) \wedge [\![\text{页}_i]\!]^{\uparrow} \qquad \Gamma_i = \Gamma \mathbin{⩓} (t_i /\!\!/ \text{程序}_i) \\ \begin{cases} t_i' = \bot \text{ 且 自由变量}(e_i) \subseteq \text{定义域}(\Sigma) \cup \text{定义域}(\Gamma_i) & \text{如果 } t_i \le \bot \\ \Sigma \cdot \Gamma_i \vdash e_i : t_i' & \text{否则} \end{cases}\end{array}}{\Sigma \cdot \Gamma \vdash \text{对 } e \text{ 进行 case 匹配 } (pg_i \to e_i)_{i \in I} : \bigvee_{i \in I} t_i'}$$

d-pair
$$\frac{\Sigma \cdot \Gamma \vdash e_i : t_i \quad (i = 1, 2)}{\Sigma \cdot \Gamma \vdash (e_1, e_2) : t_1 \times t_2}$$

D-SUB
$$\frac{\Sigma \cdot \Gamma \vdash e : t' \qquad t' \le t}{\Sigma \cdot \Gamma \vdash e : t}$$

$\boxed{\Sigma \vdash \text{定义 确定}}$ 类型定义

D-定义NO-注释
$$\frac{\Sigma(x) = t \qquad \Sigma \cdot \emptyset \vdash \lambda y.e : t}{\Sigma \vdash x = \lambda y_{\circ}\ e \text{ 确定}}$$

D-定义-a NNOT
$$\frac{\Sigma(x) = \sigma \qquad \text{ftv}(\sigma) = \emptyset \qquad \sigma = \forall A \,.\, \bigwedge_{i \in I} (t_i' \to t_i) \qquad (\forall i \in I) \quad \Sigma \cdot \{y : t_i'\} \vdash e : t_i}{\Sigma \vdash x : \ \sigma = \lambda y_{\circ}\ e \text{ 确定}}$$

$\boxed{\text{环境}(\text{定义}) = \Sigma;\ \Sigma}$ 由定义得到的环境

D-ENVS-NO-注释
$$\frac{A = \text{ftv}(t)}{\text{环境}(x = e) = \{x : t\};\ \{x : \forall A \circ t\}}$$

D-ENVS-注解
$$\frac{\sigma = \forall A \,.\, t \qquad A = \text{ftv}(t)}{\text{环境}(x : \sigma = e) = \{x : \sigma\};\ \{x : \sigma\}}$$

$\boxed{\vdash \text{程序} : t}$ 程序类型推导

D-PROG
$$\frac{(\forall i \in I)\ \text{环境}(\text{定义}_i) = \Sigma_i;\ \Sigma_i' \qquad \Sigma_{i \in I} \vdash \text{定义}_i \text{ ok} \qquad \Sigma'_{i \in I} \cdot \emptyset \vdash e : t}{\vdash \text{letrec定义}_{i \in I} \text{ 在 } e \text{ 中} : t}$$

图10。 λ_{erl}的类型规则

最后一个前提通过为 i第 个分支指定输出类型 t_i'来检查该分支的函数体。整个 `case`表达式的类型随后为所有输出类型的并集。如果输入类型 t_i 是底类型，则该分支永远无法匹配，因此输出类型 t_i' 也为底类型。针对交类型检查函数时，跳过无法匹配的分支至关重要。否则，即使某个分支对交类型中的特定部分永远无法匹配，该分支中的类型错误仍会被报告。下文将给出一个示例。[5] 如果输入类型 t_i 不是底类型，那么我们就在细化后的环境 Γ_i中检查该分支的函数体 e_i 。

辅助判断 $\Sigma \vdash$ 定义 ok 会在方案环境 Σ下检查定义 *def* 的正确性。一般而言，交类型的类型重建是不可判定的 [47]，因此我们的系统从不猜测这类类型。因此，规则 D-DEF-NO-ANNOT 针对没有类型注解的定义，只检查定义右侧是否具有环境中预期的类型。规则D-DEF-ANNOT 支持注解中的交类型，方法是针对交集中的每种类型分别检查函数体。由于 Σ 只包含顶层函数的绑定，因此类型方案 σ 必须是封闭的。

规则 D-PROG 对整个程序进行类型检查。对于每个定义 *defi*，它会计算两个方案环境 Σ_i; Σ_i'。对于带有类型注解的定义，两个环境完全相同。但对于不带类型注解的定义， Σ_i 仅包含一个单态类型，因为带有多态递归的类型重建是不可判定的 [25]。随后，这些可能是单态的类型 $\Sigma_{i\in I}$ 用于对定义进行类型检查，而泛化$\Sigma'_{i\in I}$ 则用于对主表达式进行类型检查 e。

3.3.3 示例

我们通过概述如何对第2.3节中的 `filtermap` 示例进行类型检查，来结束对 λ_{erl} 类型规则的说明。我们为`filtermap`提出了如下精确类型：

```
-spec filtermap(函数((T) -> 布尔值()), [T]) -> [T]; (函数((T) -> {true , U} | false), [T]) -> [U]; (函数((T) -> {true , U} | 布尔值()), [T]) -> [T | U].
```

现在我们展示如何在 λ_{erl}中编码`filtermap` 。列表表示为嵌套的对，并以单例整数 0 作为终止符。因此，列表的类型满足递归方程列表 $\alpha = 0 \vee (\alpha \times$ 列表 $\alpha)$。整数 0 和 1 分别表示布尔值 false 和 true，因此*bool*是类型 $0 \vee 1$ 的缩写。

```
letrec
 filtermap
   : ∀α, β.(  (α → bool) → list α → list α
            ∧ (α → ((1 × β) ∨ 0)) → list α → list β
            ∧ (α → ((1 × β) ∨ bool)) → list α → list (α ∨ β))
   = λf l. case l of 0 → 0
                     (x, l′) →
                       case f x of 0 → filtermap f l′
                                   1 → (x, filtermap f l′)
                                   (1, y) → (y, filtermap f l′)
in ...
```

[5]如果实现检测到某个分支无法匹配交集的所有变体，或者根本不存在交集，就会报告错误。

对*filtermap* 进行类型检查时，会针对交集中的三种箭头类型分别验证其函数体。这里，我们重点讨论第二种箭头类型：

$$(\alpha \to ((1 \times \beta) \vee 0)) \to \mathit{list}\,\alpha \to \mathit{list}\,\beta$$

初始环境为 $\Gamma_0 = \{ f\colon \alpha \to ((1 \times \beta) \vee 0),\ l\colon \text{列表}\ \alpha\}$。检查外层 case 表达式的第一个子句很简单。对第二个子句进行模式匹配后，我们得到环境 $\Gamma = \Gamma_0 \cup \{x\colon \alpha,\ l'\colon \text{列表}\ \alpha\}$。

我们展示规则 D-CASE 如何在此环境中对内层 case 表达式进行类型检查。审查表达式 $e = f\ x$ 的类型为 $t = (1 \times \beta) \vee 0$。内层 case 中三个模式的潜在类型和接受类型如下：

$$\begin{aligned} [\![0]\!]^{\downarrow} = [\![0]\!]^{\uparrow} &= 0 \\ [\![1]\!]^{\downarrow} = [\![1]\!]^{\uparrow} &= 1 \\ [\![(1, y)]\!]^{\downarrow} = [\![(1, y)]\!]^{\uparrow} &= 1 \times \top \end{aligned}$$

因此，$t \le 0 \vee 1 \vee 1 \times \top$，所以 case 的各个分支是穷尽的。各分支的输入类型如下：

$$\begin{aligned} t_1 &= t \wedge 0 && = 0 \\ t_2 &= (t \setminus 0) \wedge 1 && = \bot \\ t_3 &= (t \setminus (0 \vee 1)) \wedge (1 \times \top) = (1 \times \beta) \wedge (1 \times \top)) && = 1 \times \beta \end{aligned}$$

我们看到 $t_2 = \bot$，这与我们的直觉一致：针对交集的第二个变体进行类型检查时，永远不会执行第二个分支。因此，输出类型 $t_2' = \bot$。对于另外两个输出类型，我们有 $t_1' = \text{列表}\,\beta = t_3'$。这里，第三个分支是在扩展环境 $\Gamma,\ y\colon \beta$ 下进行检查的。

最后，整个 case 表达式的类型为

$$t_1' \vee t_2' \vee t_3' = \mathit{list}\,\beta \vee \bot \vee \mathit{list}\,\beta = \mathit{list}\,\beta$$

符合要求。

3.4 类型可靠性

类型可靠性是任何类型系统的一项重要性质。我们的系统以 Castagna 等人的多态变体类型系统为基础。 [12], 该系统已被证明是可靠的。我们采用 Wright 和 Felleisen 常用的句法方法来证明类型可靠性。 [50]完整证明请参见第A节。

定理 3.1（程序的进展性）。设 prog = letrec Δ in e 是一个程序，且 $\vdash \mathit{prog} : t$ 对某个 t成立。那么， e 要么是一个值，要么存在一个表达式 e'，使得 $\vdash$ letrec Δ in e letrec Δin e'。 $\rightsquigarrow$

定理 3.2（程序主体归约）。设 prog 是一个程序， t 是一个类型，且 $\vdash \mathit{prog} : t$。若 $\vdash \mathit{prog}\ \mathit{prog}'$ 对某个程序prog'成立，则 $\vdash \mathit{prog}' : t$。

推论3.3（类型可靠性）。设程序 = letrec Δ in e 满足$\vdash \mathit{prog} : t$，其中 t为某个类型。那么，prog 要么发散，要么归约为 letrec Δ in v，其中 v 是某个值，且 $\vdash$ letrec Δin $v : t$。

3.5 算法化类型规则

图10 中的类型规则无法直接用于类型推导算法，因为这些规则不是语法制导的。问题出在规则 D-SUB：它可以应用于任何表达式形式，将其类型提升为某个超类型。因此，我们将这些类型规则重新表述为一组语法制导的约束生成规则。随后简化生成的约束，得到一组子类型约束。最后， [10] 的 tally 算法求解这组子类型约束。

3.5.1 约束

图 11 定义了约束 C 和程序约束 P的语法。约束的语法大多是标准的，见 [38, 第 10]章。子类型约束 $t_1 \leq t_2$ 要求类型 t_1 是类型 t_2的子类型；变量约束 $x \leq t$ 要求变量x 的类型是 t的子类型；定义约束 def Γ in C 为约束 C定义 Γ 中的变量；两个约束 $C_1 \,\&\, C_2$ 的合取要求约束 C_1 和 C_2都可满足。由 case 表达式产生的约束需要进一步说明。这种约束的形式为 case C of $(\Gamma_i$ in C_i unless $\hat{}\, C_i)_{i \in I}$。这里， C 包含审查表达式的约束， Γ_i 引入模式匹配为分支 $\hat{}\, i$产生的变量，而 C_i 表示该分支主体的约束。约束 C_i 表达了可以执行第 $\hat{}\, i$个分支的条件：如果 C_i 可满足，那么该分支不可达。换句话说：要么必须满足 C_i，因为该分支可以执行；要么必须满足 C_i，因为该分支不可达。

程序约束 P 的形式为 letrec $C; \Sigma$ in C'。这里， C 是顶层函数的约束，其结果为模式环境 Σ； C' 是主表达式的约束。

在某些地方，我们用 True 作为约束 $\bot \leq \top$的缩写。

3.5.2 约束生成

判断 $e : t \Rightarrow C$ 为具有类型 t的表达式 e 生成约束 C（图 11）。变量 c-var、常量 c-const、 λ表达式 c-abs、应用 c-app 和对 c-pair 的规则都很直接。c-case 规则处理 case 表达式。在此规则中，约束 C 是审查表达式 e的约束。约束 D 收集 C，以及用于检查穷尽性的约束和所有由将带守卫模式 pg_i 与 t_i匹配而产生的约束 C_i。对于第 i个分支， t_i 是输入类型， Γ_i 是环境，而 D_i 是主体的约束（参见规则 d-case 图 10 中的 t_i 和 Γ_i）。此外， u_i 是那些确定不匹配第 i个分支的值的类型，因为这些值要么具有接受类型 $[\![\mathrm{pg}_j]\!]^{\downarrow}$，因而匹配某个前序分支，要么不具有潜在类型 $[\![\mathrm{pg}_i]\!]^{\uparrow}$，因而无法匹配第 i个分支。因此，如果 α （审查表达式的类型变量）是 u_i的子类型，那么该分支就无法执行。于是，该分支的 unless 条件变为 $\alpha \leq u_i$。

辅助判断 $t \,/\!/\, pg \Rightarrow C; \Gamma$ 是图 8 中 $t /\!/ pg = \Gamma$ 的约束对应形式。它计算将带守卫的模式 pg 与类型为 t 的值进行匹配后得到的环境 Γ，其中约束 C 限制了新引入的类型变量。规则 C-PG-ENV 定义了这一判断；对变量 $x \in \mathrm{fv}(pg)$ 的约束 $x \leq \top$ 确保所有变量 x 都在作用域内。

判断 定义 $\Rightarrow \langle C; \Sigma \rangle$ 生成约束 C 和方案环境 Σ 对于定义定义 (参见 $\Sigma \vdash$ 定义 正确 和 环境集合(定义) ，见图 10)。最后，程序： $t \Rightarrow P$ 生成一个

约束 $C,\ D ::= t \dot{\leq} t \mid x \dot{\leq} t \mid$ 定义 Γ 于 $C \mid C \,\&\, C$
$\mid$ 情况 C 关于 $(\Gamma_i$ 于 C_i 除非 $C_i)_{i \in I}$

程序约束 $P ::= \text{letrec } C; \Sigma \text{ in } C$

$\boxed{e : t \Rightarrow C}$ 表达式的约束生成

c-var
$$x : t \Rightarrow x \dot{\leq} t$$

c-常量
$$\kappa : t \Rightarrow \text{类型}(\kappa) \dot{\leq} t$$

C-ABS
$$\frac{e : \beta \Rightarrow C \qquad \alpha,\ \beta \text{ 新鲜}}{\lambda x.e : t \Rightarrow \ (\text{定义 } \{x : \alpha\} \text{ 在 } C) \,\&\, \alpha \to \beta \dot{\leq} t}$$

C-APP
$$\frac{e_1 : \alpha \to \beta \Rightarrow C_1 \qquad e_2 : \alpha \Rightarrow C_2 \qquad \alpha,\ \beta \text{ 新鲜}}{e_1\, e_2 : \ t \Rightarrow C_1 \,\&\, C_2 \,\&\, \beta \dot{\leq} t}$$

C-对
$$\frac{e_1 : \ \alpha_1 \Rightarrow C_1 \qquad e_2 : \ \alpha_2 \Rightarrow C_2 \qquad \alpha_1,\ \alpha_2 \text{ 新鲜}}{(e_1, e_2) : t \Rightarrow \ C_1 \,\&\, C_2 \,\&\, \alpha_1 \times \alpha_2 \dot{\leq} t}$$

c-分类
$$\frac{\begin{array}{l} e : \alpha \Rightarrow C \qquad D = C \,\&\, (\alpha \dot{\leq} \bigvee_{i \in I} [\![\text{页}_i]\!]^{\downarrow}) \,\&\, \&_{i \in I}\, C_i \\ \alpha,\ \beta \text{ 新鲜} \qquad (\forall i \in I) \quad t_i = (\alpha \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow} \\ t_i \mathbin{/\!\!/} pg_i \Rightarrow \ C_i; \Gamma_i \qquad e_i : \beta \Rightarrow D_i \qquad u_i = \bigvee_{j<i} [\![pg_j]\!]^{\downarrow} \vee \neg [\![pg_i]\!]^{\uparrow} \end{array}}{\begin{array}{l} \text{case } e \text{ of } (pg_i \to e_i)_{i \in I} : \ t \\ \Rightarrow (\text{case } \ D \text{ of } (\Gamma_i \text{ in } \ D_i \text{ unless } \ \alpha \dot{\leq} u_i)_{i \in I}) \,\&\, \ \beta \dot{\leq} t \end{array}}$$

$\boxed{t \mathbin{/\!\!/} pg \ \Rightarrow C; \ \Gamma \qquad t \mathbin{/\!\!/} p \Rightarrow C; \ \Gamma}$ 模式环境

C-PG-ENV
$$\frac{t \mathbin{/\!\!/} p \Rightarrow C; \Gamma \qquad C' = \&_{x \in \text{fv}(p \text{ 当 } g)} \ (x \dot{\leq} \top)}{t \mathbin{/\!\!/} (p \text{ 当 } \ g) \ \Rightarrow C \,\&\, \ C'; \ \Gamma \mathbin{\Cap} \text{环境}(g)}$$

c-val-env
$$t \mathbin{/\!\!/} v \Rightarrow \text{True}; \emptyset$$

c-wild-环境
$$t \mathbin{/\!\!/} \_ \Rightarrow \text{True}; \emptyset$$

c-capture-e NV
$$t \mathbin{/\!\!/} \$x \Rightarrow \text{True}; \emptyset$$

c-var-e NV
$$t \mathbin{/\!\!/} x \Rightarrow \text{True}; \{x : \ t\}$$

c-对-e NV
$$\frac{\alpha_1, \alpha_2 \text{ 新鲜} \qquad \alpha_1 \mathbin{/\!\!/} p_1 \Rightarrow C_1; \ \Gamma_1 \qquad \alpha_2 \mathbin{/\!\!/} p_2 \Rightarrow C_2; \ \Gamma_2}{t \mathbin{/\!\!/} (p_1, p_2) \ \Rightarrow C_1 \,\&\, \ C_2 \,\&\, \ t \dot{\leq} (\alpha_1 \times \alpha_2); \Gamma_1 \mathbin{\Cap} \Gamma_2}$$

$\boxed{\text{定义} \ \Rightarrow \ C; \ \Sigma}$ 定义的约束生成

C-def-a NNOT
$$\frac{\text{ftv}(\sigma) \ = \emptyset \qquad \sigma = \forall A' \,.\, \bigwedge_{i \in I} (t_i' \to t_i) \qquad (\forall i \in I) \quad e : t_i \Rightarrow C_i}{x : \sigma = \lambda y.e \Rightarrow \ \ \&_{i \in I} (\text{定义 } \{y : \ t_i'\} \text{ in } C_i); \{x : \ \sigma\}}$$

c-定义-no-annot
$$\frac{\alpha \text{ 新鲜} \qquad \lambda y.e : \ \alpha \Rightarrow C}{x = \lambda y.e \Rightarrow \ \ C; \{x : \ \alpha\}}$$

$\boxed{\text{程序} : t \Rightarrow P}$ 程序的约束生成

C-PROG
$$\frac{(\forall i \in I) \ \text{定义}_i \Rightarrow \ C_i; \ \Sigma_i \qquad \Sigma = \Sigma_{i \in I} \qquad C = \&_{i \in I}\, C_i \qquad e : t \Rightarrow C'}{\text{letrec 定义}_{i \in I} \text{ 中 } \ e : \ \ t \Rightarrow \ \text{letrec } C; \Sigma \text{ in } \ C'}$$

图11. 约束生成

约束 P 来自具有期望类型 t的程序 程序 。通常，我们将 t 设为某个新鲜类型变量。

3.5.3 简单约束与约束求解

简单约束要么是子类型约束 $\lesssim$，要么是合取 &，要么是析取 + （见图 12）。约束的析取并非常规形式。它们是对 $\hat{}$ 选择进行编码所必需的 C_i 除非 C_i 在 情况下 约束。

性质 3.4（计数算法）。存在一个会终止的算法 tally，对于每个不包含析取的简单约束 c，tally(c) 都是有限的类型替换集合，其中 tally(c) 中的每个替换都是 c 的一个解。

直观地说，tally 之于子类型约束，犹如语法统一之于相等约束。然而，子类型约束集合并没有唯一解。因此，tally 返回一个解集；如果不存在解，该集合则为空。详情请参阅 Castagna 等人 [13, 性质 A.46]。

算法 tally* 扩展了 tally，使其支持析取。

定义 *3.5*（析取的计数算法）。给定某个可能包含析取的简单约束 c，tally*(c) 是一个类型替换集合。我们按如下方式计算 tally*(c)：

(1) 将 c 化为析取范式 $c_1 + 。 。 + c_n$，使得其中没有任何 c_i 包含析取。(2) 使用性质 tally 算法求解每个 c_i，得到替换集合 S_i，分别对应 $i = 1, ..., n$。(3) 定义tally*(c) = $\cup_{i=1, ..., n} S_i$。

中的每个替换tally*(c) 都是 c 的一个解，并且比其他解更一般。参见性质B.21，位于第B节。

3.5.4 约束重写

约束重写（也在图 12 中定义）会将约束 C 转换为简单约束c。表达式的约束重写规则形式为 $\Sigma \cdot \Gamma \vdash C \quad c$。合取约束和子类型约束重写为其自身。对于变量约束 $x \lesssim t$，根据环境是否包含 x，有两条规则 RC-VAR 和 RC-VAR-POLY（参见图 10中的规则 D-VAR 和D-VAR-POLY）。规则 RC-DEF 在扩展环境下重写定义约束。最后，规则 RC-CASE 重写 case 约束 case C of (Γ_i in C_i unless $\hat{} C_i)_{i \in I}$。对于每个分支 i，它分别将 C_i 和 C_i 重写为 c_i 和 $\hat{} c_i$。（重写 C_i 不需要环境 Γ_i。）接着，析取 $\hat{} c_i + c_i$ 表示 c_i 和 $\hat{} c_i$ 中只有一个必须成立。[6]

对于程序，我们使用判断 $\vdash P \quad c$。给定形式为 letrec $C; \Sigma$ in C' 的程序约束，我们首先将 C 重写为 c。然后，我们使用 tally* 求解简单约束 c。对于某个类型代换 $\theta \in$ tally*(c)，我们将 $\Sigma\theta$ 中的类型泛化，并在此环境下将 C' 重写为 c'。这一步具有非确定性，因为 tally*(c) 是一个类型代换集合 。因此，整个程序的约束不仅包含 c'，还包含equiv(θ)，以确保解与 θ 一致。

[6]如果顶层函数的类型注释包含交类型，但交集的所有变体都满足$\hat{c_i}$，或者顶层函数的类型注释不包含交类型，但满足$\hat{c_i}$，我们的实现就会报告“分支不可达”错误。

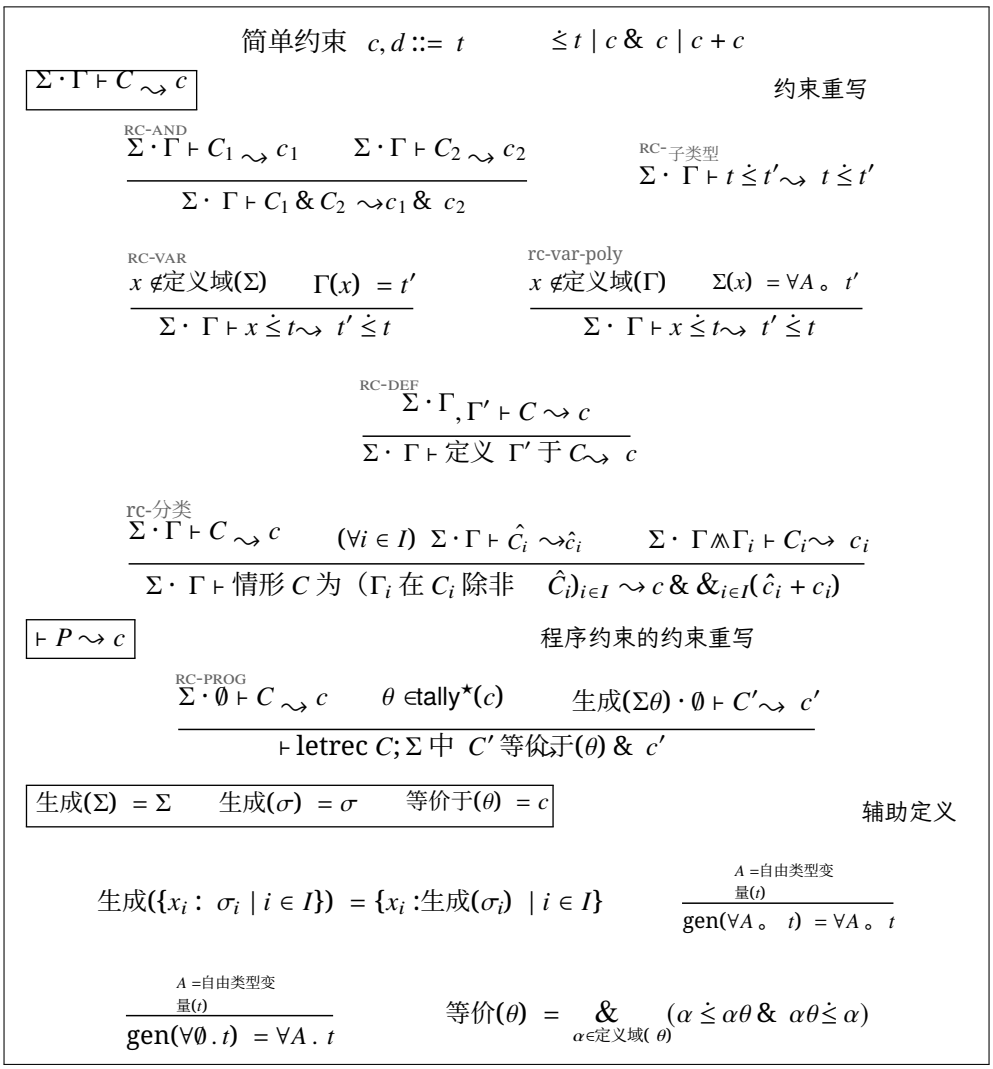

图12. 约束重写

3.5.5 程序类型检查

现在我们描述如何对程序*prog*进行类型检查。

(1) 使用 *prog* 生成程序约束 P： $\alpha \Rightarrow P$，其中 α 是某个新鲜类型变量。(2) 使用 $\vdash P \quad c$ 将 P 改写为简单约束 c。在此步骤中，规则RC-PROG 的前提可能会从tally* 返回的类型替换集合中任意选取一个 θ。(3) 使用 tally* 求解所得的简单约束 c。如果 $\text{tally}^\star(c) \neq \emptyset$，则 *prog* 类型正确。否则，我们需要返回步骤 2，在那里选取另一个类型替换θ。如果没有剩余的类型替换，则*prog* 无法通过类型检查。

3.5.6 可靠性与完备性

刚才给出的类型检查算法相对于声明式类型系统（图 10）是可靠且完备的。完整证明请参见第B 节。下文中，$\theta \Vdash c$ 表示类型替换 θ 是简单约束 c的一个解。

定理 3.6（程序算法化类型推导的可靠性）。给定程序 *prog*、某个类型 t、程序约束 P、简单约束 c以及某个类型替换 θ。假设 $prog : t \Rightarrow P$ 和 $\vdash P \quad c$ 以及 $\theta \Vdash c$。则 $\vdash prog : t\theta$。 $\rightsquigarrow$

定理 3.7（程序算法类型判定的完备性）。给定程序 *prog* 和某个类型 t。如果 $\vdash prog : t$，则存在程序约束 P、简单约束c和类型替换 θ，使得 $prog : t \Rightarrow P$ 且 $\vdash P \quad c$ 以及 $\theta \Vdash c$。 $\rightsquigarrow$

4 实现与评估

我们通过实现*Etylizer*[7],这一用 Erlang 编写的静态类型检查器，验证了我们的方法。Etylizer 的实现建立在第3节所开发的类型推导算法之上，并扩展该算法以处理多元数函数、任意大小的元组、记录、位串、列表、映射和二进制模式匹配。本节将使用真实世界的 Erlang 代码评估该工具。目前，我们将评估范围限制在顺序 Erlang 上，因为 Etylizer 尚不支持消息传递原语或行为。

我们首先介绍所采用的方法（第4.1节）。接下来，我们展示将 Etylizer 应用于 Erlang 标准库中的五个模块后得出的主要发现（第4.2节）。然后，我们考察在 Etylizer 自身代码库以及一个开源 Erlang 项目上运行 Etylizer 的结果（第4.3节）。接着，我们讨论工具的性能（第4.4节）。最后，我们将其与其他 Erlang 类型分析工具进行比较，并以此作结（第4.5节）。

4.1 案例研究：方法

我们从三个来源选取代码，以评估 Etylizer 的不同方面。为了展示其适用于经过充分测试且符合 Erlang 惯用风格的代码，我们使用 Etylizer 对 Erlang 标准库中的五个模块进行了类型检查。为了对类型系统的表达能力，以及它在针对类型检查器局限性预先编写的代码上的性能进行压力测试，我们检查了 Etylizer 自身的 14 个模块。最后，我们对开源 JSON 库 JSONE[8] 进行类型检查，以便在一个我们无法控制的代码库上提供独立的基准测试和测试。

4.1.1 模块选择

我们从 Erlang 标准库（`stdlib-7.0.2`）中选择模块时，遵循了两项标准。首先，我们选取了实现数据结构的纯函数式模块，因为它们属于我们所支持的功能范围。其次，我们将选择范围限定在缺少类型规范的顶层函数少于 20 个的模块，以确保补充缺失注解所需的工作量在可控范围内。

对于 JSONE 库，我们分析了整个项目。其规模小巧，且所有顶层函数都统一使用类型规范，因此是一个合适的案例研究。

[7]https://github.com/etylizer/etylizer[8]
https://github.com/sile/jsone

对于我们自己的 Etylizer 编译器，我们选取了一个独立的核心组件，它能够代表该工具的功能。我们有意排除了依赖于类型检查器尚未支持的语言特性的部分，例如 try/catch 表达式。

原则上，Etylizer 可以为没有类型规范的顶层函数重建类型，但这样做会带来显著的性能开销，而我们希望避免这种开销。由于一组子类型约束可能存在多个彼此不兼容的解，因此该类型系统没有主类型。因此，类型检查器首先需要重建所有未注释函数的类型，然后确定其余函数在这些解中的任意一个下是否类型安全，这可能需要进行回溯。

尽管如此，Etylizer 仍会为局部定义的函数执行类型重建，因为 Erlang 没有用于为此类函数标注类型的语法。为保持效率，局部函数推断出的类型不会进行泛化。关于 Haskell 类型检查的实证研究表明，在实践中，局部函数的类型泛化很少是必需的 [48]。

4.1.2 类型检查器执行

在本次评估中，Etylizer 在单独的一次运行中处理每个模块。我们对每个函数的处理结果进行了分类：

- **类型安全：** Etylizer 已成功验证函数体是否符合其类型规范。
- **类型错误：** 类型检查器发现了问题。这可能意味着函数在接收有效输入时可能崩溃、返回超出其指定类型范围的值，或者对于一个执行安全的函数，类型系统无法验证其类型安全性。由类型系统局限性导致的情况会被明确标识并讨论；理想情况下，不应出现此类情况。
- **待处理：** 结果既不能明确判定为通过，也不能确定为失败。对该函数进行类型检查需要一项 Etylizer 尚未实现的功能。
- **超时：** 求解子类型约束未能在预设时限内完成；对于非常复杂的表达式（例如，较长的常量列表、大型 case 表达式），有时会出现这种情况。

4.1.3 代码修改

在某些情况下，将现有代码交给 Etylizer 之前，我们必须先对其进行调整。我们进行了两类修改：添加缺失的类型规范，以及指出哪些函数子句有意不满足穷尽性要求。除此之外，我们没有进行其他类型的代码修改。

添加类型规范。 现代 Erlang 代码通常会考虑类型来编写，导出函数包含类型规范也是一种常见做法。案例研究中使用的 Etylizer 代码库和 JSONE 库中的所有顶层函数都已添加此类规范。相比之下，Erlang 标准库经常省略私有辅助函数的规范。正如第 4.1.1 节所述，出于性能考虑，我们选择为所有顶层函数添加类型规范，并将顶层函数的高效类型重建和渐进式类型检查留待未来工作。总计，我们添加了 47 个类型规范。

穷尽性违规。对于有意使用非穷尽模式的情况，我们会针对特定函数禁用穷尽性检查，而不是添加无实际作用的全匹配子句。这

模块	LOC	函数			待处理	avg	最大值	
有序集合	265	23	21	2	0	0,9s	9s	20s
队列	750	50	47	3	0	1,5s	39s	75s
有序字典	300	23	19	4	0	1,1s	5s	26s
属性列表	500	35	17	0	18	0,9s	2s	32s
sets	1000	55	36	9	10	1,1s	8s	62s

表1。Erlang 标准库模块（版本 7.0.2）的评估结果。前三列分别列出模块名称、代码行数以及模块中的函数数量。接下来的几列显示 Etylizer 将多少个函数归类为“类型安全”“类型错误”或“待处理”（参见第 4.1.2 节）。两列“平均值”和“最大值”分别显示对一个函数进行类型检查所需的平均时间和最长时间。最后几列显示对该模块进行类型检查所需的总时间。

此方法适用于两种情况：函数或`case` 表达式有意采用非穷尽形式，或者由于数据不变量，模式在语义上是穷尽的，尽管它们在语法上并未覆盖所有情况。例如，Erlang 标准库中的`orddict:fetch/2` 会在键值对列表中查找特定的键。它的设计是在传入空列表时崩溃。我们没有让其类型规范变得复杂，也没有添加冗余子句，而是仅针对这些函数通过自定义标志禁用穷尽性模式匹配检查。这样既保留了原本简单的规范，又为类型检查器提供了正式指令，同时也为阅读或修改该模块的开发者提供了明确的内联文档。

4.2 评估：*Erlang* 标准库

表1 总结了 Etylizer 对 Erlang 标准库中的五个模块进行评估所得的结果。

ordsets.erl。 Erlang 的`ordsets` 是元素按顺序排序存储的集合。该模块包含 265 行代码和 23 个函数，类型检查每个函数平均约需一秒。补充两个缺失的类型注解后，21 个函数通过了类型安全验证，但发现了两个问题。具体而言，`is_set/1` 未能穷尽地处理其类型规范所涵盖的情况，因此对于规范所涵盖的某些输入会发生运行时崩溃；而`filtermap/2` 的类型规范中有一个拼写错误：使用了原子`boolean` ，而非类型`boolean()`。

queue.erl。 实现队列的 Erlang 模块约有 750 行代码，包含 50 个函数。我们对三个函数禁用了穷尽性检查。为了对队列模块中的两个函数进行类型检查，我们必须为`lists:reverse/1`定义一种覆盖（Etylizer 使用的替代类型规范）。在 Erlang 中，`lists:reverse/1` 的类型为 `[term()] -> [term()]`。通过覆盖，我们添加了 `nonempty_list(T) -> nonempty_list(T)` 这一情况，使 Etylizer 知道`reverse` 会保留列表的非空性，并将基本情况修改为类型 `[A] -> [A]` ，以保留输入输出关系。

我们为辅助函数补上了 11 个缺失的类型注解，50 个函数中有 47 个在 80 秒内成功通过类型检查，并发现了三个问题。类型

规范对于 `get/2` 而言不够精确，无法对其实现进行类型检查。它对输入和输出使用通用的列表类型 和 `term()` 类型，导致函数`get/1` 和`head/1` 出现传递性类型错误：应用函数`get/2` 时会丢失类型信息。

orddict.erl。`orddict` 模块将有序字典实现为有序键值列表，包含 23 个函数，代码约 300 行。添加两条缺失的类型规范后，我们成功对其中 19 个函数进行了类型检查，平均每个函数的验证时间约为一秒。我们对两个函数禁用了穷尽性检查：`fetch/2` 和 `update/3`。此外，对于以常量索引调用 `list:ukeysort/2` 的情况，我们定义了覆盖 `(1, [{K, V}]) -> [{K, V}]` 。

对于函数 `append/3`、`append_list/3`、`update_counter/3` 和 `merge/3`，原始类型规范有误。这些是专为处理 `orddict` 类型实例而设计的专用函数，但其类型规范的写法却暗示它们可以处理任意多态字典。

proplists.erl。该`proplists`模块实现了属性列表操作工具。其底层数据结构使用列表，并以不同元数的元组作为元素。该模块包含 35 个函数，代码约 500 行。其中 17 个函数可在约 30 秒内完成类型安全验证；另有 18 个因 Erlang 类型系统在元组类型方面的限制而被归为待处理项（参见第 4.2.1节）。

sets.erl. sets 模块实现了版本 1（基于哈希表）和版本 2（基于映射）的集合表示，包含约 1,000 行代码中的 55 个函数。其中 36 个函数在约六分钟内成功通过验证；由于 Erlang 类型系统的限制（同样是元组类型问题，参见第 4.2.1 节），有 8 个函数无法通过验证；另有两个函数因 Etylizer 尚不支持映射生成器推导式而无法验证，因此共有 10 个函数被归入待处理类别。此外，我们还发现了 9 个问题。函数 `update_bucket/3`、`get_slot/2`、`get_bucket/2`、`maybe_expand/1`、`maybe_expand_segs/1`、`maybe_contract/2` 和 `maybe_contract_segs/1` 被规范为可处理这两种集合数据结构版本，但其实现却假定了特定版本。类型检查器正确识别了这一不匹配。此外，函数 `filtermap/2` 产生了类型错误，因为它使用`new([{version, 1}])`初始化一个表达式，而该表达式的类型为 `set(none())`。理想情况下，`new/0` 应具有多态类型 `new() -> set(A)` ，从而可作为任意元素类型 `A`的空集；但 Erlang 编译器会将仅出现在返回类型中的变量视为无界变量，并将其判定为编译错误。最后，`is_element/2`、`add_element/2` 和 `del_element/2` 的类型错误均为传递性错误，源于它们所调用函数中的类型规范不精确。

4.2.1 类型错误分析

接下来，我们将讨论对标准库模块进行类型检查时遇到的四类不同问题。每一类都代表一种不同的软件质量问题。

不精确的类型规范。我们发现了 17 处违规情况，其中类型规范过于不精确，无法准确反映函数语义。一个具有代表性的例子出现在

sets.erl 中，该模块支持两种内部实现，但类型规范未能将二者区分开来：

```
-opaque set(Element) :: #set{segs :: segs(Element)} % 基于记
录| #{Element => ?VALUE}. % 基于映射

-spec get_slot(set(E), E) -> non_neg_integer(). get_slot(T, Key) ->H =
erlang:phash(Key , T#set.maxn), ... % 若 T 是映射则会崩溃
```

这里，get_slot/2 需要一个 #set{} 记录，但规范允许记录和映射两种表示形式。尽管周围的代码可能会确保正确使用，Etylizer 仍会将其标记为潜在的运行时错误。这一发现表明，若将规范细化为分别描述这两种实现类型，类型系统便能正式验证预期的使用模式。

最近在函数ordsets:filtermap/2中又引入了一个无效规范示例：[9]

```
-doc(#{since => <<"OTP␣27.0">>}). -spec filtermap(Fun, Ordset1) -> Ordset2 when Fun ::
fun((Element1 :: T1) -> 布尔值 | ({true , Element2 :: T2})), Ordset1 :: ordset(T1), Ordset2 ::
ordset(T1 | T2).
```

即使不检查函数体，规范也揭示了一个明显的问题：回调函数的返回类型包含原子boolean，而不是类型boolean()。特定原子与布尔类型之间这种细微的（语法上的）区别，会被更严格的工具标记出来，这说明此类错误本可以在纳入标准库之前被发现。

实现缺陷。我们发现ordsets.erl 模块中有一个函数存在真正的错误，可能导致运行时崩溃。有序集合由有序列表表示。函数is_set/1 用于检查某个值是否确实是这样的有序集合。

```
-spec is_set(Ordset) ->布尔值() when Ordset :: term(). is_set([E |Es]) ->
is_set(Es, E); is_set([]) ->True; is_set(_) ->false.
```

```
-spec is_set(列表(term()), term()) -> >布尔值(). is_set([E2 |Es] ,
E1) when E1 <E2 -> > is_set(Es, E2); is_set([|], _) -> > false;
is_set([] ,_) -> > true.
```

主函数 is_set/1 适用于所有项，并且为非列表输入正确地定义了一个兜底函数子句。辅助函数 is_set/2 仅适用于列表。问题在于，is_set/1的输入可能是非空的不正确列表。[10] 例如，调用is_set/1 并传入参数 [1|1] 会导致is_set/2崩溃。

其他任何类型检查器都没有检测到这个错误，包括 Dialyzer、Gradualizer、eqWAlizer，以及 Valliappan 和 Hughes 的类型推断算法 [46]。

[9]提交 be1d1f71a18796f083aaa2a852dd01e653083186 位于 https://github.com/erlang/otp[10]不正确列表是指尾部不是列表的列表。

参数多态。一类值得注意的规格问题涉及错误的参数性：类型变量暗示的适用范围比实现实际支持的范围更广。以下是 orddict 模块中的一个示例，该模块使用经过排序的键值列表来实现字典。函数 `append_list` 会将一个值列表追加到与某个键关联的当前值列表中。

```
-spec append_list(Key, 值列表, Orddict) -> Orddict when 值列表 :: 值,
Orddict :: 有序字典(Key, 值). append_list (...) -> ...; append_list(Key,
NewList, {_K,Old}Dict) -> %键 K{Key,Old NewList}Dict;

...
```

类型规范表明，`Orddict` 的值类型可以是任意`Value`类型。正如 ++运算符所示，实现实际上要求该值的类型为`ValList`。以下是更正后的类型规范：

```
-spec append_list(键, 值列表, 有序字典) -> 有序字典 when 值列表 :: [值], 有序字典
:: 有序字典 ( 键 , 值列表 ).
```

此处，`orddict` 的值类型是`ValList`，而之前是 `Value`。使用此类型规范，Etylizer 便能够对该函数进行类型检查。

惯用崩溃行为。一些函数应用会使用惯用的 Erlang 模式，有意只实现“顺利路径”。例如：

```
-spec filelib:ensure_dir(term()) -> ok | {错误 , term()}。

-spec mkdirs(file:filename()) -> ok | {error, string()}. mkdirs(D) ->
ok = filelib:ensure_dir(filename:join(D, "temp")).
```

这里，对`ok`的模式匹配明确假定操作成功，并将任何错误结果都视为足以导致崩溃的异常。默认情况下，我们还会检查`A = B` 表达式的穷尽性。

当 Etylizer 报告非穷尽性时，需要评估以下事项：

(1) 开发者必须检查缺失的情况究竟是真正的错误，还是有意为之的设计。(2) 如果这是一个错误（缺少错误处理），则构成应当修复的类型错误。(3) 如果这是有意为之（遇到错误时提前崩溃），开发者可以通过自定义标志，为特定函数禁用穷尽性检查。（目前，无法针对单个表达式禁用穷尽性检查。）

在我们的案例研究中，我们采用了这一推理过程，并对崩溃行为显然是有意为之的函数禁用了穷尽性检查。

元组类型。对于`proplists.erl` 模块，我们的分析发现，我们的类型理论可以支持 18 个函数，但由于当前元组类型规范的限制，最终产生了类型错误。

Erlang 中的属性列表是一种列表，其中每个元素要么是非空元组，要么是原子（即元组`{A, true}`的简写）。每个元组的第一个元素用作键。（列表中可能还包含其他元素，但这些元素会被忽略。）以下是用于搜索指定键的函数`lookup/2`：

项目	mods	LOC	funs	type safe	type 错误	待处理	超时	avg	max	总计
jsone	4	1045	65	34	9	20	2	8,0s	189s	502s
etylizer	14	2770	231	204	6	13	8	8,2s	235s	1830年代
标准库	5	2815	186	140	18	28	0	1,2s	39s	215s

表 2. 案例研究：ErlangProjects。

```
-spec lookup(term(), [term()]) ->none | tuple(). lookup(Key, [P
  | Ps]) ->if is_atom(P), P =:= Key -> {Key , true}; tuple_size(P)
  >= 1, element(1, P) =:= Key ->P; true -> lookup(Key , Ps) end;
lookup(_Key , []) -> none.
```

给定的类型规范 `term()` 对于`P` 而言过于宽泛，无法对实现进行类型检查。相反，这种模式要求指定一个元组在第一个位置包含键的类型的元素，而这一概念目前无法用 Erlang 的类型语言表达。同样，`sets.erl` 模块中被归类为待处理的函数，由于依赖于处理大小不一的元组，也无法得到验证。版本 1 的实现大量使用元组，将哈希段用作动态数组。缺少这一特性并不构成根本性限制，因为集合论类型原则上能够表达此类属性。

4.2.2 Erlang 标准库中的更多模块

要检查标准库中规模更大、复杂度更高的模块，需要处理更多语言特性，尤其是 `try` 表达式；此外，还需要支持渐进式类型，或提升类型重建的性能。目前，我们正在按 [6], 所述引入渐进类型，并提升类型重建的性能。

4.3 评估：*Erlang* 项目

表 2 展示了在 Etylizer 自身代码和开源库 JSONE 的部分代码上运行 Etylizer 的结果。为求完备，表中还总结了 Erlang 标准库的结果，我们已在上一节中进行了详细讨论。

我们选取了 Etylizer 自身的一些核心模块，共 14 个模块，包含 231 个函数，总计 2770 行代码。对这些模块进行类型检查后，发现 6 个类型错误；8 个函数导致超时。对于 13 个函数，由于存在尚未实现且不受支持的表达式（try-catch、名义类型），Etylizer 无法判断其类型安全性。此外，出于与第 4.1.3 节所述类似的原因，我们对 Etylizer 的部分代码禁用了穷尽性检查，共涉及 3 个函数。

对于 JSONE 项目，我们纳入了全部四个模块，共计 65 个函数。这里我们遇到了 9 个类型错误，还有 20 个待处理。Etylizer 在两个函数上超时。

代码修改。 除了第 4.1.3 节中提到的修改外，在将代码传递给 Etylizer 之前，我们还必须对其进行一些额外调整。(1) 类型检查器不支持记录字段的默认值，因此我们必须显式地为此类字段传递值。(2) 有一个函数使用异构列表，仅仅是为了对第一个元素进行特殊处理。

% everywhere(F, X) 递归地将 *F* 应用于 *X*。*% F* 是一个函数，当 *X* 应递归遍历时返回 *% * {rec , X}*，当 *X* 不应再进行处理时返回 *% * {ok, X}*，当列表、元组和映射应执行默认遍历时返回 *% * error*。*% X* 是要遍历的值。-spec everywhere(fun((term()) ->{rec , term()} | {ok, ter m()} | error , term()) ->term(). everywhere(F, T) ->TransList =fun(L) -> lists:map(fun(X) -> everywhere(F, X) end , L) end , case F(T) of error ->case T of X when is_list(X) ->TransList(X); X when is_tuple(X) -> list_to_tuple(TransList(tuple_to_list(X))); X when is_map(X) -> maps:from_list(TransList(maps:to_list(X))); X ->X end; {ok, X} ->X; {rec, X} -> everywhere(F, X) end.

图 13. 需要二阶多态的 everywhere 函数。

我们改写了函数头，使其将第一个元素作为额外参数接受。(3) 我们修正了一处错误的类型规范。(4) 从 Erlang 项存储中检索值时，只能得到 `term()` 类型，因此类型信息会丢失。因此，我们必须为一个函数添加守卫，以便在进一步处理之前恢复预期的类型。

4.3.1 类型错误

秩-*2* 类型。 评估发现，Etylizer 中有一个函数无法进行类型检查：一个通用遍历函数，它会递归转换任意嵌套数据结构（图 13）。

该函数的灵感来自 Haskell 库“scrap your boilerplate”中的通用 `everywhere` 遍历。在 Haskell 中对 `everywhere` 函数进行类型检查需要二阶多态 [28]。我们的类型系统缺少这一特性，因此这是唯一一个无法通过改进类型检查器来解决的类型错误。

在符合 Erlang 惯用风格的代码中，这种模式并不常见；开发者通常会针对每种数据结构编写特定类型的遍历，而不是使用通用转换。

重叠类型。 Etylizer 代码库中的所有其他类型错误都源自重叠的带标签联合类型。请看 `form()` 的规范，其中元组的第三个元素用作区分并集成员的标识符：

-type import() :: {attribute , anno(), import , {atom(), [带元数的函数]}}. -type wild_attr() :: {attribute , anno(), atom(), term()}. -type form() :: import() | wild_attr() | ...

要通过 case 表达式将 `form()` 细化为 **`import()`**，我们需要根据 **`import()`**类型的结构进行模式匹配：

```
case Form of% Form 是 Forms 列表中的一个元素
  F ={attribute , Anno, import , {ModName , 函数}} ->...
```

因此，我们预期在匹配的分支中，`F` 的类型为**`import()`**。然而，`wild_attr` 类型会阻止唯一匹配，因为它的第三个分量 `atom()` 与**`import()`** 类型中更具体的 **`import`** 原子重叠。元组的第四个分量也是如此。

除了更改数据结构本身，一个优雅的解决方案是扩展类型语言，加入类型差集。这样便可以从`wild_attr`的第三个位置排除**`import`** 原子，使用类似 `ety:without(atom(), import())`的规范。这样无需更改数据结构，就能消除重叠。不过，为了保持与 Erlang 标准类型语言的兼容性，我们没有更改类型语言。

精确数字类型。 由于在算术运算中跟踪整数范围存在局限，JSONE 库中的两个函数需要禁用精确数字类型验证。

```
-type utc_offset_seconds() :: -86399..86399. -spec local_offset() -> utc_offset_seconds().
local_offset() ->UTC = {{1970, 1, 2}, {0, 0, 0}}, 本地时间 =
calendar:universal_time_to_local_time ({{1970, 1, 2}, {0, 0, 0}}),
calendar:datetime_to_gregorian_seconds(Local) -
calendar:datetime_to_gregorian_seconds(UTC)。
```

减法运算会丢失输入值的精确范围信息，因此无法验证结果是否落在指定界限内。不过，狭窄的返回类型对于文档编写和 API 清晰度仍然很有价值。Etylizer 支持通过一个标志，选择性地对特定函数禁用精确数字检查，同时保留精确规格；这体现了对这样一种情况的认可：实现依靠外部保证来维持范围不变量，而非通过静态验证来确保这一点。

依赖类型。 Erlang 中有些操作本质上是动态的，要精确描述其类型就需要值依赖类型。下面给出两个示例，以及它们在 Erlang 标准库中的类型。

- `erlang:element/2`，类型为 `(pos_integer(), tuple()) -> term()`，用于访问元组中的第 N 个元素。
- `erlang:setelement/3`，类型为 `(pos_integer(),tuple(),term())->tuple()`，用于设置元组中的第 N 个元素。

若要给出更精确的类型，就需要在类型层级反映元组的长度和元素类型。由于我们无法修改标准库中的官方类型规范，因此采用类型覆盖。借助这种方式，我们可以通过重载这些函数，为静态已知的索引（单例类型）和特定元组形状定义相应的情形，从而细化这类函数的类型。

```
% 元素覆盖-spec 'erlang:element' (1, {A}) -> A; '
erlang:element ' (2, {_, B}) -> B; '
erlang:element ' (2, {_, B, _}) -> B; ...;
```

```
(_, tuple()) ->term().
```

这些类型覆盖编码了带有专门分支的函数的语义行为，实际上是在类型层级表达模式匹配。

为了评估依赖值的类型系统的实际影响，我们对六个 Erlang 项目进行了实证分析。结果表明，尽管像 `element/2`和 `setelement/3` 这样的函数本质上是动态的，但其大部分用法（1820 次调用中的 76%）使用的是字面量常量索引。这些情况正是我们的类型覆盖所能准确处理的，它们为每个常量索引提供了规范。其余 24% 的调用使用非常量表达式，属于真正的动态情况，需要完整的依赖类型。（完整研究请参见第D节。）

这表明，尽管理论上依赖值的类型系统颇具挑战性，但它对验证典型 Erlang 代码的实际影响有限，因为我们的方法足以处理常见情况。

对于依赖运行时的索引，目前采用单例类型进行重载的方法并不足够。未来可以扩展类型语言，例如引入元组元数多态（比如提供一种类型，如`tuple(T)`，表示任意大小且所有元素类型均为`T`的元组），从而验证更多动态情况。例如，JSONE 库中的`hex/1` 函数会对一个由整数构成的同质元组进行索引；届时便可以验证该函数。

```
-spec 'erlang:element' (整数(), 元组(T)) -> T. -spec hex(字节()) -
> 0..16#FFFF. hex(X) ->element(X + 1, {16#3030, 16#3031, ...}).
```

目前，这种特定模式仍未经验证，被归类为待处理事项，并留待未来处理。

4.4 性能与超时

Etylizer 提供了一套富有表现力且强大的类型系统。它支持多种特性，例如模式匹配的冗余检查和穷尽性检查、有限形式的类型收窄、具有并集类型和交类型的多态函数，以及局部函数和顶层函数的类型重建（尽管真实场景的案例研究仅依赖局部函数的类型重建）。这能为程序行为提供细粒度的静态保证，并使现有的 Erlang 代码片段也能够通过类型检查。

这种表现力的计算成本相当高。集合论类型系统中的子类型判定是 EXPTIME 完全 [14]，；虽然 CDuce [3] 展现了实用的类型检查性能，但 Etylizer 还面临着为局部定义的函数进行类型重建所带来的额外复杂性。如何在集合论类型和多态类型并存的情况下，大幅提升这类包含局部定义函数类型重建的约束系统的性能，仍是一项尚待解决的挑战。

虽然平均验证时间仍只有几秒，但某些函数需要数分钟才能完成类型检查。通过缓存验证结果，可以减轻性能影响，确保每个函数在开发过程中只需承担一次这项开销。分布显示，大多数函数都能快速完成验证，只有少数特别复杂的案例需要较长的验证时间。

总的来说，在整个案例研究中，有 2% 的函数无法通过 Etylizer 的类型检查，并且超出了 5 分钟的超时限制。这些函数大多会对深度嵌套、递归定义的类型进行递归变换，例如 Erlang 抽象语法树的类型，因此会产生包含 40 多个分支的 case 表达式。提升这类函数的性能，是进一步推动集合论类型发展的一项关键挑战。

4.5 与其他静态类型分析工具的比较

Erlang 有多种静态类型检查器和分析器，例如 Dialyzer[41],eqWAlizer [49],以及 [24]。虽然每种工具各有特点，但它们的类型系统都存在一些局限。例如，考虑并集与积类型的分配律：

```
-spec dist_left({ok   | err, nil}) ->{ok, nil}   | {err , nil}. dist_left(X) ->X.

-spec dist({ok   |  err , arg   |  nil}) ->{ok, arg}   |  {err, arg}   |
{ok, nil}   |  {err, nil}. dist(X) ->X.
```

T积类型对并集的分配律指出，对于任意类型 A、B 和 C:

$$(A \cup B) \times C \equiv (A \times C) \cup (B \times C) \qquad \text{(left distributivity)}$$

$$A \times (B \cup C) \equiv (A \times B) \cup (A \times C) \qquad \text{(right distributivity)}$$

单独应用一次分配律是可接受的，但由于采用了特设实现，eqWAlizer 和 Gradualizer 在嵌套应用分配律时都会产生类型错误，而 Etylizer 则通过其形式化基础处理分配律。在我们的示例中，`dist_left` 能通过这三个工具的类型检查，而`dist`只有 Etylizer 能通过类型检查。

同样，交类型也带来了挑战：

```
-spec inter (整数()) ->  整数(); (原子()) ->原子(). inter(X) ->  case X of _
when is_integer(X) ->X  +  1; _ ->X end.
```

eqWAlizer 和 Gradualizer 都无法正确处理带有交类型的分支中的出现类型，导致出现意料之外的类型错误，因而无法对该函数进行类型检查。相比之下，Etylizer 中 case 表达式、守卫和交类型之间的交互符合预期，但线性模式除外。由于 Gradualizer 已不再积极维护，它还面临其他挑战。

Dialyzer 按照其设计，会忽略某些类型不一致，不予检测。它采用的成功类型分析方法能够保证不会产生误报，但代价是可能漏掉实际存在的错误。因此，在我们的对比评估中，Dialyzer 未能报告其他工具检测到的若干类型错误。不过，需要注意的是，在所分析代码库的开发过程中，Dialyzer 已经被使用过，这意味着它本可以发现的许多问题早已得到修正。因此，Dialyzer 适合作为有用的基准，而不适合作为公平的比较对象，用于评估对未经检查代码的检测能力。Etylizer 对这些类型场景的处理更为严谨，但其更强的表达能力也带来了更高的计算成本。这些对比

观察结果得到了定量分析的证实 [42]，也得到了定性评估的支持[4]。

案例研究中检查的所有代码都是在考虑 Dialyzer 的情况下开发的，并且均通过了其分析。Gradualizer 的比较价值有限，因为它为实际上类型正确的函数报告了大量类型错误。

与 eqWAlizer 的比较揭示了更实质性的差异：Etylizer 识别出的 28 个问题中，有 14 个未能被 eqWAlizer 发现。性能特征也有显著差异：eqWAlizer 几乎可以即时提供反馈，但代价是问题检测能力有所降低。这种权衡反映了集合论验证与其他方法之间的根本差异。Etylizer 更高的计算投入带来了更强的保证，并揭示了那些逃过速度更快但精确度较低的分析的类型错误。

5 相关工作

Erlang 的类型检查器。旨在从形式上约束 Erlang 类型系统的相关类型系统研究由来已久。这项工作始于 Marlow 和 Wadler 的研究 1997年；我们以他们的类型规范系统作为基准 [2]。此后，研究进一步发展到 Dialyzer [29, 30, 41]，它展示了在该社区成功采用静态类型系统所必需的实际考量。因此，这可能需要作出某些权衡。

多种类型系统已作为研究原型开发，用于研究和探索 Erlang 环境中的各种方面与方法，例如渐进式类型 [24]、Hindley-Milner 风格的类型推断 [18] 、支持数据构造器重载的类型推断 [46], 以及双向类型 [20, 39]。然而，这些项目要么仅支持 Erlang 的一个子集，要么需要进行大量修改，因此不适合采用。每种方法都缺少提供全面解决方案所必需的关键组件。我们的系统以 [42], 的形式化为基础，整合了在实际应用中切实采用所必需的所有类型方面。

与我们的工作并行，研究人员正努力为 Elixir 引入静态类型系统。Elixir 是一种与 Erlang 密切相关的语言。该类型系统基于集合论类型 [6, 7]，与 Etylizer 类似。它提供了与我们的方法相近的功能，尤其侧重于强箭头的渐进式类型，以及对带守卫的模式进行更精细的分析。截至版本 1.17，其中一些功能已集成到 Elixir 中。[11] 我们认为，我们的工作与这些努力相辅相成。他们的方法着重于单项功能，而我们的目标是评估一种全面形式化框架及其实现的可行性，并将各个方面整合其中。此外，他们的工作开发了一种新的类型语言，而我们的类型检查器则建立在成熟的 Erlang 类型语言之上。未来一个颇有前景的方向，是将渐进类型 [6, 7] 纳入 Etylizer。这样便能够对许多函数未附带类型规范的项目进行类型检查。

集合论类型。 在本文中，我们整合并扩展了基于集合论类型的研究成果 [10, 11, 14, 23]。我们通过纳入带守卫的模式匹配来扩展这项研究，使表达式类型能够根据动态条件进行判断，从而实现类型收窄。尽管这种方法只能近似守卫所捕获的值集合，但对于模式匹配中的类型收窄和穷尽性检查来说已经足够。

[11] https://hexdocs.pm/elixir/main/gradual-set-theoretic-types.html

[9] 提出了一种充分发挥出现类型全部能力的方法。然而，在存在副作用的情况下设计和应用该系统仍面临挑战；其对性能的影响也尚未在更全面的案例研究中得到充分评估。尽管如此，定性和定量研究 [4, 42] 表明，我们的类型收窄方法对于 Erlang 中的实际应用而言足够有效且切实可行。

与 [12] 和 [35], 类似，我们通过使用计数算法求解子类型约束来进行类型检查和类型重建。这使我们能够推断局部定义的函数的类型。我们进一步扩展了他们的工作，加入了模式守卫和穷尽性检查。

为动态类型语言改造静态类型系统。有许多语言最初采用动态类型，后来加入了静态类型系统。例如，Flow 和 TypeScript 是 JavaScript 的类型扩展 [16, 40]。两者都已在实践中得到广泛应用，支持并集类型和交类型，但缺少否定类型。与 Erlang 类似，Python 提供了类型注解语法[12]，但不强制进行静态类型检查。Mypy[13] 和 Pyright[14] 等工具试图通过以临时方式推断函数参数来检测类型不一致。值得注意的是，与 Etylizer 不同，JavaScript 和 Python 的类型系统都缺乏形式化基础。

MLstruct [34] 支持记录、等递归类型，以及并集、交类型和否定类型，并以代数方式而非语义方式定义子类型关系。该语言支持具有主类型的类型推断。Etylizer 也包含一种有限形式的类型推断，但不包括交类型，并且不保证存在主类型。然而，MLstruct 的若干设计选择并不适合 Erlang。例如，尽管 Erlang 允许重载算术运算符，MLstruct 却不支持使用交类型对重载进行编码。此外，MLstruct 将记录与函数类型的并集视为等同于顶类型，缺乏区分此类并集两部分的机制。相比之下，Erlang 通过 `is_function`类型测试提供了这种区分。

Typed Racket [45] 引入了*occurrencetyping*，允许自定义类型测试，并根据变量出现的上下文细化变量类型。与我们的方法类似，变量的类型可以根据其遇到的类型测试而发生变化。在 Etylizer 中，类型测试集合遵循 Erlang 语言并且是固定的。此外，Erlang 和 Racket 都支持在其类型语言中编码精确的数值类型 [43],不过 Erlang 只提供了 Racket 中可用数值类型的一个子集。

6 结论

本文介绍了一种基于集合论类型和语义子类型的顺序 Erlang 形式化方法。通过这项工作，我们在以可行且实用的方式提升 Erlang 代码的静态类型安全性方面取得了重大进展。在理论方面，我们成功证明了类型可靠性和类型检查的可判定性。我们的系统能够生成信息丰富、易于理解的错误消息，帮助开发者更有效地诊断和解决类型相关问题。虽然我们已扩展类型检查器以支持映射和记录，但由于篇幅所限，对于 [42]，提到的其中一项局限，我们无法在此详细介绍。类型检查器的性能

[12] https://peps.python.org/pep-0484/
[13] https://mypy-lang.org/
[14] https://github.com/microsoft/pyright

已大幅提升，在某些情况下超越了 CDuce 的最先进性能，尤其是在内部约束求解器方面。我们还为局部函数实现了类型推断，并为由多个模块组成的程序实现了类型检查，显著增强了系统的易用性和表达能力。

不过，未来工作仍面临若干挑战，也存在不少机遇。目前尚未支持一些语言构造，例如 try-catch 和不透明类型。此外，随着 Erlang 不断演进，maybe 表达式和映射推导式等新构造也陆续出现，这些构造有望从使用集合论类型进行精确类型检查中获益良多。还可以进一步改进某些语言构造和比较运算符的类型细化，以提高准确性和表达能力。虽然现在已支持局部函数的类型推断，但非局部函数的类型推断仍然缓慢，需要进一步研究以优化性能。最后，尽管 tally 算法已变得更加高效，求解大型约束集合仍是一项挑战，还需要开展更多研究来提升其可扩展性和效率。

较大型程序通常会逐步迁移到静态类型检查，这意味着代码库中的某些部分经过静态检查，而其他部分仍采用动态类型。鉴于我们取得的进展，按照 [6]所述的渐进式类型将是我们系统的一项有价值的扩展。渐进式类型允许静态类型代码与动态类型代码无缝共存，这尤其有用，因为模块通常既包含导出且带类型的函数，也包含私有的无类型函数。对于程序中那些本质上难以进行静态类型检查的部分，例如动态加载的代码或尚未得到全面支持的构造，它也可以提供实用的解决方案。

利益冲突

无。

参考文献

[1] Mart´ın Abadi 和 Marcelo P. Fiore。1996年。递归类型的语法考量。载于《计算机科学逻辑》第11届 IEEE 年会论文集，美国新泽西州新不伦瑞克，1996年7月27日至30日。IEEE 计算机学会，242–252。doi:10.1109/LICS.1996.561324[2] Joe Armstrong。2007年。Erlang 的历史。载于第三届 ACM SIGPLAN 编程语言历史会议（HOPL-III）论文集，美国加利福尼亚州圣迭戈，2007年6月9日至10日，Barbara G. Ryder 和 Brent Hailpern（编）。ACM，1–26。doi:10.1145/1238844.1238850
[3] V´eronique Benzaken、Giuseppe Castagna 和 Alain Frisch。2003年。CDuce：一种以 XML 为中心的通用语言。载于第八届 ACM SIGPLAN 国际函数式编程会议论文集（ICFP 2003），瑞典乌普萨拉，2003年8月25日至29日，Colin Runciman 和 Olin Shivers（编）。ACM，51–63。doi:10.1145/944705.944711[4] Florian Berger、Albert Schimpf、Annette Bieniusa 和 Stefan Wehr。2024年。同中有异：Erlang 静态类型检查器的比较分析。载于第23届 ACM SIGPLAN Erlang国际研讨会论文集（Erlang 2024），意大利米兰，2024年9月2日。ACM。doi:10.1145/3677995. 3678189[5] Richard S. Bird 和 Lambert G. L. T. Meertens。1998年。嵌套数据类型。载于《程序构造数学》（MPC'98）会议论文集（计算机科学讲义，第1422卷），瑞典马尔斯特兰德，1998年6月15日至17日，Johan Jeuring（编）。Springer，52–67。doi:10.1007/BFb0054285[6] Giuseppe Castagna 和 Guillaume Duboc。2024年。守卫分析与安全擦除渐进式类型：一种面向 Elixir 的类型系统。arXiv:2408.14345 [cs.PL] https://arxiv.org/abs/2408.14345[7] Giuseppe Castagna、Guillaume Duboc 和 Jos´e Valim。2023年。Elixir 类型系统的设计原则。《艺术、科学与工程编程》8，2（2023）。doi:10.22152/PROGRAMMING-JOURNAL.ORG/2024/8/4

[8] Giuseppe Castagna 和 Alain Frisch。2005年。语义子类型温和入门。载于第七届 ACM SIGPLAN 声明式编程原理与实践国际会议论文集，2005年7月11日至13日，葡萄牙里斯本，Pedro Barahona 和 Amy P. Felty（编）。ACM，198–199。doi:10.1145/ 1069774.1069793[9] Giuseppe Castagna、Micka¨el Laurent 和 Kim Nguyen。2024年。动态语言的多态类型推断。ACM编程语言会议录 8，POPL（2024），1179–1210。doi:10.1145/3632882[10] Giuseppe Castagna、Kim Nguyen、Zhiwu Xu 和 Pietro Abate。2015年。具有集合论类型的多态函数：第二部分：局部类型推断与类型重建。载于第42届 ACM SIGPLAN-SIGACT 编程语言原理年会论文集（POPL 2015），印度孟买，2015年1月15日至17日，Sriram K. Rajamani 和 David Walker（编）。ACM，289–302。doi:10.1145/2676726.2676991[11] Giuseppe Castagna、Kim Nguyen、Zhiwu Xu、Hyeonseung Im、Sergue¨ı Lenglet 和 Luca Padovani。2014年。具有集合论类型的多态函数：第一部分：语法、语义与求值。载于 POPL 会议论文集。ACM，5–18。doi:10.1145/2535838.2535840[12] Giuseppe Castagna、Tommaso Petrucciani 和 Kim Nguyen。2016年。多态变体的集合论类型。载于第21届 ACM SIGPLAN 函数式编程国际会议论文集（ICFP 2016），日本奈良，2016年9月18日至22日，Jacques Garrigue、Gabriele Keller 和 Eijiro Sumii（编）。ACM，378–391。doi:10.1145/2951913.2951928[13] Giuseppe Castagna、Tommaso Petrucciani 和 Kim Nguyen。2016年。多态变体的集合论类型。CoRR abs/1606.01106（2016）。arXiv:1606.01106 http://arxiv.org/abs/1606.01106 含证明的扩展版本。[14] Giuseppe Castagna 和 Zhiwu Xu。2011年。参数多态与子类型关系的集合论基础。载于 SIGPLAN 会议论文集。ACM，94–106。doi:10.1145/2034773.2034788[15] CDuce 2025。CDuce 集合论类型检查器。https://gitlab.math.univ-paris-diderot.fr/cduce/cduce/[16] Avik Chaudhuri、Panagiotis Vekris、Sam Goldman、Marshall Roch 和 Gabriel Levi。2017年。面向 JavaScript 的快速精准类型检查。ACM编程语言会议录 1，OOPSLA（2017），48:1–48:30。doi:10.1145/3133872[17] Karl Crary、Robert Harper 和 Sidd Puri。1999年。什么是递归模块？载于 SIGPLAN 会议论文集。ACM，50–63。doi:10.1145/301618.301641[18] Lu´ıs Damas 和 Robin Milner。1982年。函数式程序的主类型方案。载于第九届 ACM 编程语言原理年会会议记录，美国新墨西哥州阿尔伯克基，1982年1月。ACM 出版社。doi:10.1145/582153.582176[19] Flemming M. Damm。1994年。并集类型、交类型与递归类型的子类型关系。载于 TACS 会议论文集（LNCS，第789卷）。Springer，687–706。doi:10.1007/3-540-57887-0 121[20] Jana Dunfield 和 Neel Krishnaswami。2022年。双向类型。《ACM 计算调查》54，5（2022）。doi:10.1145/3450952[21] Ericsson AB。2025年。类型与函数规范（摘自《Erlang 系统文档，v28.1.1》）。https://www.erlang.org/doc/system/typespec.html[22] Alain Frisch。2004年。XML 适用编程语言的理论、设计与实现。博士论文。[23] Alain Frisch、Giuseppe Castagna 和 V´eronique Benzaken。2008年。语义子类型：以集合论方式处理函数类型、并集类型、交类型与否定类型。《ACM 学报》55，4（2008），19:1–19:64。doi:10.1145/1391289.1391293[24] Gradualizer 2022。https://github.com/josefs/Gradualizer。[25] Fritz Henglein。1993年。多态递归的类型推断。《ACM 程序设计语言系统汇刊》15，2（1993），253–289。doi:10.1145/169701.169692[26] Ralf Hinze。2000年。完美树与位反转置换。《函数式编程杂志》10，3（2000），305–317。doi:10.1017/s0956796800003701[27] Miguel Jimenez、Tobias Lindahl 和 Konstantinos Sagonas。2007年。一种用于指定 Erlang 类型契约及其与成功类型分析交互的语言。载于2007年 SIGPLAN Erlang 专题研讨会论文集（德国弗赖堡）（ERLANG '07）。计算机协会，美国纽约州纽约，11–17。doi:10.1145/1292520.1292523[28] Ralf L¨ammel 和 Simon L. Peyton Jones。2003年。抛弃样板代码：一种实用的泛型编程设计模式。载于 TLDI'03 会议论文集：2003年 ACM SIGPLAN 语言设计与实现中的类型国际研讨会，美国路易斯安那州新奥尔良，2003年1月18日，Zhong Shao 和 Peter Lee（编）。ACM，26–37。doi:10.1145/604174.604179

[29] Tobias Lindahl 和 Konstantinos Sagonas。2004年。通过轻量级静态分析检测电信应用中的软件缺陷：一个实战故事。载于《编程语言与系统：第二届亚洲研讨会，APLAS 2004》，台北，台湾，2004年11月4日至6日。论文集（《计算机科学讲义》，第3302卷），Wei-Ngan Chin（编）。Springer，91–106。doi:10.1007/978-3-540-30477-7 7[30] Tobias Lindahl 和 Konstantinos Sagonas。2006年。基于成功类型的实用型推断。载于 SIGPLAN 论文集。ACM，167–178。doi:10.1145/1140335.1140356[31] Simon Marlow 和 Philip Wadler。1997年。一种实用的 Erlang 子类型系统。载于1997年 ACM SIGPLAN国际函数式编程会议（ICFP '97）论文集，阿姆斯特丹，荷兰，1997年6月9日至11日，Simon L. Peyton Jones、Mads Tofte 和 A. Michael Berman（编）。ACM，136–149。doi:10.1145/258948.258962[32] Dimitris Mostrous 和 Vasco Thudichum Vasconcelos。2011年。轻量级 Erlang 的会话类型检查。载于《协调模型与语言——第13届国际会议，COORDINATION 2011》，雷克雅未克，冰岛，2011年6月6日至9日。论文集（《计算机科学讲义》，第6721卷），Wolfgang De Meuter 和 Gruia-Catalin Roman（编）。Springer，95–109。doi:10.1007/978-3-642-21464-6 7[33] Fabian Muehlboeck 和 Ross Tate。2018年。通过集成子类型关系赋能并集类型和交类型。《ACM编程语言会议录》2，OOPSLA（2018）。doi:10.1145/3276482[34] Lionel Parreaux 和 Chun Yin Chau。2022年。MLstruct：结构类型布尔代数中的主类型推断。《ACM编程语言会议录》6，OOPSLA2（2022）。doi:10.1145/3563304[35] Tommaso Petrucciani。2019年。函数式语言的多态集合论类型。（多态集合论类型用于函数式语言）。博士论文。索邦巴黎城，法国。doi:10.15167/ PETRUCCIANI-TOMMASO PHD2019-03-14[36] Benjamin Pierce。1992年。使用交类型和有界多态进行编程。博士论文。美国。[37] Benjamin Pierce。2002年。《类型与编程语言》（第1版）。麻省理工学院出版社。[38] Benjamin C. Pierce。2004年。《类型与编程语言高级主题》。麻省理工学院出版社。[39] Nithin Vadukkumchery Rajendrakumar 和 Annette Bieniusa。2021年。Erlang 的双向类型检查。载于第20届 ACM SIGPLAN Erlang国际研讨会论文集，Erlang@ICFP 2021，线上活动，韩国，2021年8月26日，Stavros Aronis 和 Annette Bieniusa（编）。ACM，54–63。doi:10.1145/ 3471871.3472966[40] Aseem Rastogi、Nikhil Swamy、C´edric Fournet、Gavin M. Bierman 和 Panagiotis Vekris。2015年。TypeScript 的安全高效渐进式类型。载于第42届 ACM SIGPLAN-SIGACT 编程语言原理年会论文集，POPL 2015，孟买，印度，2015年1月15日至17日，Sriram K. Rajamani 和 David Walker（编）。ACM，167–180。doi:10.1145/2676726.2676971[41] Konstantinos Sagonas。2005年。开发 Dialyzer 的经验：一种用于检测 Erlang 应用程序缺陷的静态分析工具。载于 ACM软件缺陷检测工具评估研讨会。[42] Albert Schimpf、Stefan Wehr 和 Annette Bieniusa。2022年。Erlang 的集合论类型。载于 IFL 2022 论文集。ACM。doi:10.1145/3587216.3587220[43] Vincent St-Amour、Sam Tobin-Hochstadt、Matthew Flatt 和 Matthias Felleisen。2012年。数值塔的类型检查。载于《声明式语言的实践方面——第14届国际研讨会，PADL 2012》，费城，宾夕法尼亚州，美国，2012年1月23日至24日。论文集（《计算机科学讲义》，第7149卷），Claudio V. Russo 和 Neng-Fa Zhou（编）。Springer，289–303。doi:10.1007/978-3-642-27694-1 21[44] Josef Svenningsson。2024年。josefs/Gradualizer。https://github.com/josefs/Gradualizer 原始日期：2017-10- 29T19:30:20Z。[45] Sam Tobin-Hochstadt 和 Matthias Felleisen。2008年。带类型的Scheme的设计与实现。载于 POPL 论文集。ACM，395–406。doi:10.1145/1328438.1328486[46] Nachiappan Valliappan 和 John Hughes。2018年。Erlang 中对动态代码进行类型检查。载于 SIGPLAN 论文集。ACM，49–60。doi:10.1145/3239332.3242766[47] Steffen van Bakel。1995年。交集类型赋值系统。《理论计算机科学》151，2（1995），385–435。doi:10.1016/0304-3975(95)00073-6[48] Dimitrios Vytiniotis、Simon L. Peyton Jones、Tom Schrijvers 和 Martin Sulzmann。2011年。OutsideIn(X)：具有局部假设的模块化类型推断。《函数式编程杂志》21，4-5（2011）。doi:10.1017/S0956796811000098[49] WhatsApp。2024年。eqWAlizer：Erlang 类型检查器。https://github.com/WhatsApp/eqwalizer/[50] Andrew K. Wright 和 Matthias Felleisen。1994年。类型可靠性的语法方法。《信息与计算》115，1（1994）。doi:10.1006/INCO.1994.1093

类型安全：声明式系统

本附录证明第3 节中定义的声明式类型系统是可靠的。

定义 *A.1*（自由类型变量）。类型 t中的自由类型变量，记作 $\mathsf{ftv}(t)$，以余归纳方式定义为 $\mathsf{ftv}_0(t,\ \emptyset)$：

$$\mathsf{ftv}_0(t,\mathcal{T}) = \begin{cases} \emptyset & \text{if } t \in \mathcal{T} \\ \mathsf{ftv}_1(t,\mathcal{T}\cup\{t\}) & \text{otherwise} \end{cases}$$

$$\begin{aligned} \mathsf{ftv}_1(\alpha,\mathcal{T}) &= \{\alpha\} \\ \mathsf{ftv}_1(b,\mathcal{T}) &= \emptyset \\ \mathsf{ftv}_1(t_1\times t_2,\mathcal{T}) &= \mathsf{ftv}_0(t_1,\mathcal{T})\cup\mathsf{ftv}_0(t_2,\mathcal{T}) \\ \mathsf{ftv}_1(t_1\to t_2,\mathcal{T}) &= \mathsf{ftv}_0(t_1,\mathcal{T})\cup\mathsf{ftv}_0(t_2,\mathcal{T}) \\ \mathsf{ftv}_1(t_1\vee t_2,\mathcal{T}) &= \mathsf{ftv}_1(t_1,\mathcal{T})\cup\mathsf{ftv}_1(t_2,\mathcal{T}) \\ \mathsf{ftv}_1(\neg t_1,\mathcal{T}) &= \mathsf{ftv}_1(t_1,\mathcal{T}) \end{aligned}$$

- 某个类型方案中的自由类型变量定义为 $\mathsf{ftv}(\forall A\,.\,t) = \mathsf{ftv}(t)\setminus A$。
- 某个类型替换中的自由类型变量为 $\mathsf{ftv}([t_i/\alpha_i \mid i\in I]) = \cup_{i\in I}\mathsf{ftv}(t_i)$。
- 我们以显而易见的方式将 ftv 的定义扩展到其他构造。
- 如果一个类型 t 满足$\mathsf{ftv}(t)$为空，我们就称它为封闭类型。

性质 A.2（积类型的投影）。对于某个类型 $t \le \top\times\top$，存在投影 $\pi_1(t)$ 和 $\pi_2(t)$，使得 *(i)* $t \le \pi_1(t)\times\pi_2(t)$; *(ii)* 如果 $t \le t_1\times t_2$，那么 $\pi_i(t) \le t_i$; *(iii)* 如果 $t \le t' \le \top\times\top$，那么 $\pi_i(t) \le \pi_i(t')$; *(iv)* 对所有类型替换 θ，$\pi_i(t\theta) \le \pi_i(t)\theta$。参见 *Castagna* 等人 *[13,* 性质 *A.31*，第 *26]* 页。

性质 A.3（子类型关系的性质）。对于类型 t_1，t_2：(*i*) $\bot \le t \le \top$ 对任意 t。(*ii*) 如果 $t_1\to t_2 \le t_1'\to t_2'$，那么 $t_1' \le t_1$ 且 $t_2 \le t_2'$。(*iii*) 如果 $t_1\times t_2 \le t_1'\times t_2'$，那么 $t_1 \le t_1'$ 且 $t_2 \le t_2'$。(*iv*) 如果 $t_1 \le t_2$，那么对于任意类型替换 θ，$t_1\theta \le t_2\theta$。

证明。 断言 (i)、(ii) 和 (iii) 可由类型作为值集合的解释推出，见 [23]。关于 (iv)，见 [14, 引理 3.15，第 10] 页。

定义 *A.4*（绑定模式变量）。由模式绑定的表达式变量，记作 $\mathsf{pbv}(p)$，定义如下：

$$\mathsf{pbv}(p) = \begin{cases} \{x\} & \text{if } p = x \\ \mathsf{pbv}(p_1)\cup\mathsf{pbv}(p_2) & \text{if } p = (p_1,p_2) \\ \emptyset & \text{otherwise} \end{cases}$$

由带守卫模式绑定的表达式变量为$\mathsf{pbv}(p \text{ when } g) = \mathsf{pbv}(p)$。

定义 *A.5*（自由表达式变量）。某个表达式 e 中的自由表达式变量，记作$\mathsf{fv_e}(e)$，定义如下：

$$\mathsf{fv_e}(x) = \{x\}$$
$$\mathsf{fv_e}(\kappa) = \emptyset$$
$$\mathsf{fv_e}(\lambda x\,.\,e) = \mathsf{fv_e}(e) \setminus \{x\}$$
$$\mathsf{fv_e}(e_1\ e_2) = \mathsf{fv_e}(e_1) \cup \mathsf{fv_e}(e_2)$$
$$\mathsf{fv_e}((e_1, e_2)) = \mathsf{fv_e}(e_1) \cup \mathsf{fv_e}(e_2)$$
$$\mathsf{fv_e}(\mathtt{case}\ e\ \mathtt{of}\ (pg_i \to e_i)_{i\in I}) = \mathsf{fv_e}(e) \cup \bigcup_{i\in I} (\mathsf{fv_{pg}}(pg_i) \cup (\mathsf{fv_e}(e_i) \setminus \mathsf{pbv}(pg_i)))$$

某个模式 p 中自由出现的表达式变量，记作 $\mathsf{fv_p}(p)$，定义如下

$$\mathsf{fv_p}(v) = \mathsf{fv_e}(v)$$
$$\mathsf{fv_p}(\_) = \emptyset$$
$$\mathsf{fv_p}(x) = \emptyset$$
$$\mathsf{fv_p}(\$x) = \{x\}$$
$$\mathsf{fv_p}((p_1, p_2)) = \mathsf{fv_p}(p_1) \cup \mathsf{fv_p}(p_2)$$

某个守卫 g 中自由出现的表达式变量，记作$\mathsf{fv_g}(g)$，定义如下

$$\mathsf{fv_g}(\mathtt{true}) = \emptyset$$
$$\mathsf{fv_g}(\mathtt{oracle}) = \emptyset$$
$$\mathsf{fv_g}(\mathtt{is}_{gt}(v)) = \mathsf{fv_e}(v)$$
$$\mathsf{fv_g}(\mathtt{is}_{gt}(x)) = \{x\}$$
$$\mathsf{fv_g}(g_1\ \mathtt{and}\ g_2) = \mathsf{fv_g}(g_1) \cup \mathsf{fv_g}(g_2)$$

某个带守卫模式 pg 中自由出现的表达式变量，记作 $\mathsf{fv_{pg}}(pg)$，定义为$\mathsf{fv_{pg}}(p\ \mathtt{when}\ g) = \mathsf{fv_p}(p) \cup (\mathsf{fv_g}(g) \setminus \mathsf{pbv}(p))$.
大多数情况下，我们省略 $\mathsf{fv_e}$、$\mathsf{fv_g}$、$\mathsf{fv_p}$、$\mathsf{fv_{pg}}$ 中的下标，直接写作 fv。

引理 A.6（子类型包含消去）。若 $\Sigma \cdot \Gamma \vdash e : t$，则存在一个关于 $\Sigma \cdot \Gamma \vdash e : t'$ 的推导，其末尾不以规则 *D-SUB* 结束，并且 $t' \leq t$。

证明。 通过对类型推导进行直接归纳即可。

定义 *A.7*（环境的子类型关系）。 我们用 $\Gamma \leq \Gamma'$ 表示定义域(Γ) =定义域(Γ')，且对所有 $x \in$ 定义域(Γ)， $\Gamma(x) \leq \Gamma'(x)$ 均成立。

引理 A.8（弱化）。若 $\Sigma \cdot \Gamma \vdash e : t$ 且 $\Gamma' \leq \Gamma$，则 $\Sigma \cdot \Gamma' \vdash e : t$。

证明。 通过对 $\Sigma \cdot \Gamma \vdash e : t$ 的推导进行直接归纳即可。

引理 A.9（额外变量）。若 $\Sigma \cdot \Gamma \vdash e : t$ 且 $x \notin$ 定义域(Σ) $\cup$定义域(Γ)，则 $\Sigma \cdot \Gamma,\ x : t' \vdash e : t$ 对任意 t'。

证明。 通过对 $\Sigma \cdot \Gamma \vdash e : t$ 的推导进行直接归纳即可。

引理 A.10（值形状）。设 v 是一个值。(i) 若 $\Sigma \cdot \Gamma \vdash v : \mathsf{ty}(\kappa)$ 对于某个 κ 成立，则 $v = \kappa'$ 对于某个 κ' 成立，且满足 $\mathsf{ty}(\kappa) = \mathsf{ty}(\kappa')$；并且若 κ 是一个整数，则 $\kappa = \kappa'$。

(*ii*) 若 $\Sigma \cdot \Gamma \vdash v : t_1 \to t_2$，则 v 的形式为 $\lambda x。e$，且 $\Sigma \cdot \Gamma、x : t_1 \vdash e : t_2$。
(*iii*) 若 $\Sigma \cdot \Gamma \vdash v : t_1 \times t_2$，则 v 的形式为 (v_1,v_2)，$\Sigma \cdot \Gamma \vdash v_1 : t_1$，且 $\Sigma \cdot \Gamma \vdash v_2 : t_2$。

证明。
(i) 若 $\Sigma \cdot \Gamma \vdash v : \mathsf{ty}(\kappa)$，则根据引理 A.6 $\Sigma \cdot \Gamma \vdash v : t$，最后一步应用的规则为 D-CONST，且$t \leq \mathsf{ty}(\kappa)$。
- 若 κ 是整数，则 $t = \mathsf{ty}(\kappa) = \kappa$，因此 $v = \kappa$。
- 若 κ 是浮点数，则 $t = \mathsf{float}$，因此 v 也必定是浮点常量。(ii) 若 $\Sigma \cdot \Gamma \vdash v : t_1 \to t_2$，则 $\Sigma \cdot \Gamma \vdash v : t'_1 \to {}^{t'}2$，最后一步应用的规则为 d-abs，且根据引理 A.6 有 $t'_1 \to {}^{t'}{}_2 \leq t_1 \to t_2$。于是 $v = \lambda x.e$ 并且由规则 $\Sigma \cdot \Gamma, x : t'_1 \vdash e : t'2$ 的前提可知。结合性质 A.3(ii)、引理 A.8 和规则 d-sub，我们得到 $\Sigma \cdot \Gamma, x : t_1 \vdash e : t_2$。(iii) 若 $\Sigma \cdot \Gamma \vdash v : t_1 \times t_2$，则根据性质 A.3(iii)，可类似前一种情况得出结论。

LEM引理 A.11. 假设 $\Sigma \cdot \Gamma \vdash v : t$。那么对于任意类型 u，要么 $\Sigma \cdot \Gamma \vdash v : u$，要么 $\Sigma \cdot \Gamma \vdash v : \neg u$。

证明。这可由类型作为值集合的解释，以及对于任意类型 u，u 和 $\neg u$ 的解释的并集是全部值集合这一事实得出。参见 Frisch 等人[23, 引理 6.22，第 30]页。

引理 A.12. 对于任意 Σ 和 Γ，不存在满足 $\Sigma \cdot \Gamma \vdash v : \bot$的值 v。

证明。参见 Frisch 等人 [23, 引理 6.19]。我们通过对 $\Sigma \cdot \Gamma \vdash v : t$ 的推导进行归纳，证明蕴含命题“若 $\Sigma \cdot \Gamma \vdash v : t$，则$t \neq \bot$”。如果推导以规则 D-CONST、D-ABS或D-PAIR结束，则结论显然成立。对于规则 D-SUB，根据归纳假设可得结论。由于 v 是一个值，推导不可能以其他四条规则中的任何一条结束。

引理 A.13. 若 $\Sigma \cdot \Gamma \vdash v : t$ 对某个 t 和 $v \# gt$ 成立，则 $\Sigma \cdot \Gamma \vdash v : \mathsf{ty}(gt)$。

证明。
根据 gt的形式进行分类讨论。
- 情形 gt = 整数：则 v 必须是一个整数常量，因此该命题由C-CONST得证。
- 情形 gt = 浮点数：则 v 必须是一个浮点数常量，因此该命题由 C-CONST得证。
- 情形 gt = 对：则 $v = (v_1, v_2)$。根据假设 $\Sigma \cdot \Gamma \vdash v : t$ 以及引理 A.6，则 $\Sigma \cdot \Gamma \vdash v : t_1 \times t_2$ 对某个 t_1，t_2。现在，应用规则 d-sub

$$\Sigma \cdot \Gamma \vdash v : \underbrace{\mathsf{ty}(\mathtt{pair})}_{=\top\times\top}$$

- 分类讨论 gt = 函数：与前一种情况类似。

结束分类讨论。

引理 A.14（模式匹配动态语义的全定义性）。
(i) 若 $fv(p) \subseteq dom(\Delta)$，则 $v /_\Delta p$ 有定义；也就是说，它要么是 Ω，要么是某个 η。
(ii) 若 $fv(g) \subseteq dom(\Delta)$，则 $G_\Delta(g)$ 有定义；也就是说，它要么为 $true$，要么为 $false$。
(iii) 若 $fv(pg) \subseteq dom(\Delta)$，则 $v /_\Delta pg$ 有定义；也就是说，它要么是 Ω，要么是某个 η。

证明。断言 (i) 由对 p 的归纳法得出，断言 (ii) 由对 g 的归纳法得出。断言 (iii) 由 (i) 和 (ii) 得出。

引理 A.15。

(i) 若 $v/_\Delta p = \eta$，则 $dom(\eta) = pbv(p)$。*(ii)* 若 $v/_\Delta(p\ when\ g) = \eta$，则 $dom(\eta) = pbv(p)$。*(iii)* 对任意 t，$pbv(p) = dom(t /\!\!/ p)$。*(iv)* 对任意 t，$pbv(pg) \subseteq dom(t /\!\!/ pg)$。

证明。 断言 (i) 可通过对 p 进行归纳法得出。断言 (ii) 由 (i) 得出。断言 (iii) 可通过对 p 进行归纳法得出。断言 (iv) 由 (iii) 得出。

引理 A.16。(*i*) 环境(g) 是封闭的。(*ii*) $[\![p]\!]^\rho_\Gamma$ 是封闭的，当且仅当 Γ 是封闭的。(*iii*) $[\![pg]\!]^\rho$ 是封闭的。

证明。 断言 (i) 可通过对 g 进行归纳法得出，断言 (ii) 可通过对 p 进行归纳法得出。断言 (iii) 由 (i) 和 (ii) 得出。

引理 A.17。若 $\Sigma \cdot \Gamma \vdash e : t$ 则 fv(e) $\subseteq$ 定义域(Σ) $\cup$定义域(Γ)

证明。 直接对 $\Sigma \cdot \Gamma \vdash e : t$ 进行归纳即可。

引理 A.18. 如果 $\Sigma \cdot \Gamma \vdash v : t$ 且 $t \leq \mathsf{ty}(gt)$，那么 $v\ \#gt$。

证明。 只需检查守卫类型 gt（整数、浮点数、对、函数）与值的形态（整数常量、浮点常量、对、λ 抽象）的所有可能组合即可得出。

引理 A.19. $v\ \#gt$ 当且仅当，对于所有 η，$v\eta\ \#gt$。

证明。 对 v 做归纳即可。注意，v 和 $v\eta$ 在最外层具有相同的形态。

引理 A.20. 环境($g\eta$) $= \{x : 环境(g)(x) \mid x \in 定义域(环境(g)) \setminus 定义域(\eta)\}$

证明。 对 g 做归纳即可，其中对任意值 v，环境($\mathsf{is}_{gt}(v)$) $= \emptyset$。

引理 A.21. 设 $\Gamma = t /\!\!/ pg$ 且 $\Gamma' = t /\!\!/ (pg\eta)$。则 $\Gamma' = \{x : \Gamma(x) \mid x \in 定义域(\Gamma) \setminus 定义域(\eta)\}$。

证明。 假设 $pg = p$ 当 g。不失一般性，pbv(p) 是新鲜的，因此 pbv(p) $\cap$ 定义域(η) $= \emptyset$。于是 $pg\eta = p\eta$ 当 $g\eta$。由此结论可由引理 A.20 得证。

引理 A.22. 若 $\Sigma \cdot \Gamma \vdash v : t$ 且 $\Sigma \cdot \Gamma \vdash v : t'$，则 $\Sigma \cdot \Gamma \vdash v : t \wedge t'$。

证明。 我们将该结论推广，并证明 $\Sigma \cdot \Gamma \vdash e : t$ 且 $\Sigma \cdot \Gamma \vdash e : t'$ 可推出 $\Sigma \cdot \Gamma \vdash e : t \wedge t'$。证明采用对类型推导进行归纳法，参见 Frisch 等人的 [23,引理 6.15，第 27]页。

引理 A.23。假设 $\Sigma \cdot \Gamma \vdash v : [\![p]\!]^{\downarrow 环境(g)}$。若 $v/_\Delta p = \eta$ 且安全$^{\downarrow}(g, \mathsf{pbv}(p))$，则$\mathcal{G}_\Delta(g\eta)$ =为 *True*。

证明。由 safe$^{\downarrow}(g, \mathsf{pbv}(p))$，我们知道 g 是类型测试的合取 $(\mathsf{is}_{gt_i}(x_i))_{i \in I}$ 和 $(\mathsf{is}_{gt'_j}(v_j))_{j \in J}$，满足对于所有 $i \in I$，$x_i \in$pbv(p)，并且对于所有 $j \in J$，$v_j\ \#\ gt'_j$。由引理 A.15，我们有 pbv(p) =dom(η)。结合 $v_j \#gt'_j$，根据引理 A.19，我们还有 $v_j\eta \#gt'_j$。为了证明 $G_\Delta(g\eta)$ = True，我们接下来只需证明

$$\eta(x_i)\ \#\ gt_i \text{ for all } i \in I$$

我们对 p 使用归纳法。
根据……的形式进行分类讨论 p。

- 分类讨论 $p = v$，或 $p =$ 或 $p = \$y$：要么是 $v /_\Delta p = \Omega$（不可能的情况），要么是 $v /_\Delta p = []$，因此 $I = \emptyset$。
- 分类讨论 $p = x$：于是 $\eta = [v/x]$，且对所有 $i \in I$ 都有 $x_i = x$。假设 $I \neq \emptyset$，否则该命题显然成立。根据假设 $\Sigma \cdot \Gamma \vdash v : [\![p]\!]^{\downarrow} \mathsf{env}(g)$，可得 $\Sigma \cdot \Gamma \vdash v : \mathsf{env}(g)(x)$。结合引理 A.12，$\mathsf{env}(g)(x) \neq \bot$，因此所有 gt_i 都相同，且 $\mathsf{env}(g)(x) = \mathsf{ty}(\mathsf{gt}1)$。由引理 A.18 可得 $v \# \mathsf{gt}1$。因此，对于所有 $i \in I$，$\eta(x_i) \# \mathsf{gt}_i$，这正是所需。

- 情形 $p = (p_1, p_2)$：那么 $v = (v_1, v_2)$ 并且

$$v_i /_\Delta p_i = \eta_i \quad (i = 1, 2) \tag{1}$$

$$\eta_1 \sim \eta_2$$

$$\eta = \eta_1 \cup \eta_2$$

此外，我们有 $[\![p]\!]^{\downarrow}_{\text{环境}(g)} = [\![p_1]\!]^{\downarrow}_{\text{环境}(g)} \times [\![p_2]\!]^{\downarrow}_{\text{环境}(g)}$，因此根据假设 $\Sigma \cdot \Gamma \vdash v : [\![p]\!]^{\downarrow}_{\text{环境}(g)}$ 以及引理A.10

$$\Sigma \cdot \Gamma \vdash v_i : [\![p_i]\!]^{\downarrow}_{\mathsf{env}(g)} \quad (i = 1, 2) \tag{2}$$

现在构造守卫 g_1，将类型测试 $\mathsf{is}^{gt_i}(x_i)$ 合取起来，但只取满足 $x_i \in \mathsf{pbv}(p_1)$ 的那些测试。类似地构造守卫 g_2。于是

$$\mathsf{safe}^{\downarrow}(g_i, \mathsf{pbv}(p_i)) \quad (i = 1, 2) \tag{3}$$

现在使用 (2)、(1) 和 (3) 来应用归纳假设。由此得到 $\mathsf{G}_\Delta(g_i \eta_i)$，其中 $i = 1, 2$。根据 $\mathsf{pbv}(p) = \mathsf{pbv}(p_1) \cup \mathsf{pbv}(p_2)$，我们知道每个类型测试 $\mathsf{is}^{gt_i}(x_i)$ 都包含在 g_1 或 g_2 中。根据 η 的构造，我们进而有 $\eta(x_i) \# gt_i$ 对所有 $i \in I$ 都成立。

分类讨论结束。

引理 A.24。若 $\mathcal{G}_\Delta(g\eta) =$ 为 $True$，则 $\text{安全}^{\uparrow}(g) =$ 为 $True$。

证明。对 g 进行归纳。$g = g_1$ and g_2 的情形直接由归纳假设得出，$g =$ true、$g =$ oracle 和 $g = \mathsf{is}^{gt}(x)$ 的情形则显然成立。因此，假设 $g = \mathsf{is}^{gt}(v)$。对于此情形，只有当 not $v \# gt$ 时，$\text{安全}^{\uparrow}(g) =$ 为假。由引理 A.19 可知，$v\eta \# gt$ 也不成立。因此，$\mathsf{G}_\Delta(g\eta)$ 必定为假，这就产生了矛盾。因此，$\text{安全}^{\uparrow}(g) =$ 为真，如所需。

引理 A.25。若 $\mathcal{G}_\Delta(g\eta) =$ 为 $True$，则 $\text{环境}(g) \neq \bot$。

证明。假设 $\text{环境}(g) = \bot$。因此，存在 $x \in \text{定义域}(\text{环境}(g))$，使得 $\text{环境}(g)(x) = \bot$。因此，g 至少包含两个类型测试 $\mathsf{is}^{gt_1}(x)$ 和 $\mathsf{is}^{gt_2}(x)$，其中 $gt_1 \neq gt2$，因为只有由环境合并得到的类型才可能产生 $\bot$。

- $x \in \text{定义域}(\eta)$，其中 $\eta(x) = v$。根据 $\mathsf{G}_\Delta(g\eta) = true$，可得 $v \# gt1$ 且 $v \# gt2$。这不可能，因为 $gt_1 \neq gt2$。
- $x \notin \text{定义域}(\eta)$。于是，根据 $\mathsf{G}_\Delta(g\eta) = true$，我们有 $\Delta(x) = v$，且 $v \# gt1$ 且 $v \# gt2$。这不可能，因为 $gt_1 \neq gt2$。

引理 A.26. 假设 $\Sigma \cdot \Gamma \vdash v : t$。若 $v /_\Delta p = \eta$ 且 $\mathcal{G}_\Delta(g\eta) =$ 为 $True$，且 $\text{环境}(g) \neq \bot$，则 $\Sigma \cdot \Gamma \vdash v : [\![p]\!]^{\uparrow}_{\text{环境}(g)}$。

证明。 对 p 进行归纳法。为了使归纳过程成立，我们需要将断言泛化如下：

> *Assume* $\Sigma \cdot \Gamma \vdash v : t$. *If* $v /_\Delta p = \eta$ *and* $\mathcal{G}_\Delta(g\eta') = \mathit{true}$ *for some* η' *with* $\eta \subseteq \eta'$ *and* $\mathsf{env}(g) \neq \bot$, *then* $\Sigma \cdot \Gamma \vdash v : [\![p]\!]^\uparrow_{\mathsf{env}(g)}$.

分类讨论关于 p 的形式。

- 情况 $p = v_\prime$：那么 $v = v_\prime$ 是因为 $v /_\Delta p = \eta$。如果 $v = \kappa$，那么 $[\![p]\!]^\uparrow$ 环境(g) = 类型(κ)，因此可由规则 D-CONST 得出结论。否则，$[\![p]\!]^\uparrow$ 环境$(g) = \top$，因此由于 v 类型正确，结论成立。
- 情况 $p = \_$：那么 $[\![p]\!]^\uparrow$ 环境$(g) = \top$，因此结论显而易见。
- 情况 $p = x$：那么 $\eta = [v/x]$。如果 $x \notin$ 定义域(环境(g))，那么 $[\![p]\!]^\uparrow$ 环境$(g) = \top$，因此结论显而易见。现在假设 $x \in$ 定义域(环境(g))，因此 g 至少包含一个关于 x 的类型测试。根据环境$(g) \neq \bot$，我们知道 g 中关于 x 的所有类型测试都具有 $\mathsf{is}_t(x)$ 的形式。因此，

$$[\![p]\!]^\uparrow_{\mathsf{env}(g)} = \mathsf{env}(g)(x) = \mathsf{ty}(gt)$$

  由 $\mathcal{G}_\Delta(g\eta') = \mathit{真}$ 且 $[v/x] = \eta \subseteq \eta'$，我们有 $\eta(x)\ \#gt$，因此 $v\ \#gt$。根据引理A.13，于是 $\Sigma \cdot \Gamma \vdash v : \mathsf{ty}(gt)$，如所需。
- 情形 $p = \$y$：则 $[\![p]\!]^\uparrow$ 环境$(g) = \top$，因此结论显然成立。
- 情形 $p = (p_1, p_2)$：由 $v /_\Delta p = \eta$ 可得

$$\begin{gathered} v = (v_1, v_2) \\ v_i /_\Delta p_i = \eta_i \quad (i = 1, 2) \\ \eta_1 \sim \eta_2 \\ \eta = \eta_1 \cup \eta_2 \end{gathered}$$

  由假设 $\eta \subseteq \eta'$，对于 $i = 1, 2$，我们有 $\eta_i \subseteq \eta'$。因此，应用归纳假设可得，对于 $i = 1, 2$：

$$\Sigma \cdot \Gamma \vdash v_i : [\![p_i]\!]^\uparrow_{\mathsf{env}(g)}$$

  注意到 $[\![p]\!]^\uparrow$ 环境(g) $= [\![p_1]\!]^\uparrow$ 环境(g) $\times [\![p_2]\!]^\uparrow$ 环境(g)，应用规则 D-PAIR 可得$\Sigma \cdot \Gamma \vdash v : [\![p]\!]^\uparrow$ 环境(g)，如所需。

结束分类讨论。

引理 A.27（潜在类型和接受类型的特征）。假设 $\Sigma \cdot \Gamma \vdash v : t$对于某个 t。

(*i*) 若 $\Sigma \cdot \Gamma \vdash v : [\![pg]\!]^\downarrow$，则对于某个 η，有 $v /_\Delta pg = \eta$。(*ii*) 若对于某个 η，有 $v /_\Delta pg = \eta$，则 $\Sigma \cdot \Gamma \vdash v : [\![pg]\!]^\uparrow$。(*iii*) 若 $v /_\Delta pg = \Omega$，则 $\Sigma \cdot \Gamma \vdash v : \neg[\![pg]\!]^\downarrow$。

证明。 结论 (iii) 可由 (i) 和引理 A.11 得出。

结论 *(i)* 的证明。 设 $pg = p$ 当 g。于是根据引理 A.12 $[\![pg]\!]^{\downarrow} \neq \bot$。根据定义，我们有

$$\mathsf{safe}^{\downarrow}(g, \mathsf{pbv}(p)) = \mathit{true}$$
$$\mathsf{env}(g) \neq \bot$$
$$p \text{ linear}$$
$$[\![pg]\!]^{\downarrow} = [\![p]\!]^{\downarrow}_{\mathsf{env}(g)} \neq \bot$$

我们对 p 进行归纳
分类讨论 p 的形状。

- 分类讨论 $p = v$：于是，

$$[\![pg]\!]^{\downarrow} = [\![p]\!]^{\downarrow}_{\mathsf{env}(g)} = \kappa$$
$$v' = \kappa$$
$$\mathsf{ty}(\kappa) = \kappa$$

  根据 $[\![pg]\!]^{\downarrow}$ 的定义。因此， $\Sigma \cdot \Gamma \vdash v : \kappa$ 由假设可得。由于只有整数是单例类型，κ 必须是整数，因此由引理 A.10(i) 可得 $v = \kappa$。于是，由MATCH-VAL可得 $v /_{\Delta} p = []$。根据引理 A.23 ，$\mathsf{G}_{\Delta}(g) = \mathit{True}$。因此，由MATCH-TRUE可得 $v /_{\Delta} pg = []$。
- 分类讨论 $\_p =$： 于是，由 MATCH-WILD可得 $v/_{\Delta} p = []$。再根据引理 A.23 ，$\mathsf{G}_{\Delta}(g) = \mathit{True}$，因此由 MATCH-TRUE可得该结论。
- 情况 $p = x$：根据MATCH-VAR，有 $v /_{\Delta} p = [v/x]$。再根据引理A.23，$\mathsf{G}_{\Delta}(g) = \mathit{true}$，因此由MATCH-TRUE可得结论成立。
- 情况 $p = \$y$：不可能，因为$[\![\$y]\!]^{\downarrow}_{\text{环境}(g)} = \bot$.
- 情况 $p = (p_1, p_2)$：那么，

$$[\![p]\!]^{\downarrow}_{\mathsf{env}(g)} = [\![p_1]\!]^{\downarrow}_{\mathsf{env}(g)} \times [\![p_2]\!]^{\downarrow}_{\mathsf{env}(g)}$$

  根据引理A.10(iii)，我们有

$$v = (v_1, v_2)$$
$$\Sigma \cdot \Gamma \vdash v_i : [\![p_i]\!]^{\downarrow}_{\mathsf{env}(g)} \quad i = 1, 2 \tag{1}$$

  模式 p 是线性的，因此$\mathsf{pbv}(p_1) \cap \mathsf{pbv}(p_2) = \emptyset$ 和 p_1， p_2 都是线性的。此外，由安全$^{\downarrow}(g,\ \mathsf{pbv}(p)) = \mathit{True}$，我们知道 g 仅包含针对$\mathsf{pbv}(\ p)$中变量的类型测试。

  将$\mathsf{pbv}(p_2)$ 中针对变量的类型测试替换为 true，从而构造 g_1。以类似方式构造g_2。那么对于 $i = 1,\ 2$:

$$\mathsf{safe}^{\downarrow}(g_i, \mathsf{pbv}(p_i))$$
$$\mathsf{env}(g_i) \neq \bot$$

  当 g_i 时，定义 $pg_i = p_i$。那么对于 $i = 1,\ 2$:

$$[\![pg_i]\!]^{\downarrow} = [\![p_i]\!]^{\downarrow}_{\mathsf{env}(g_i)} = [\![p_i]\!]^{\downarrow}_{\mathsf{env}(g)}$$

  由（1），可得 $i = 1,\ 2$:

$$\Sigma \cdot \Gamma \vdash v_i : [\![pg_i]\!]^{\downarrow}$$

现在应用归纳假设，可得 $v_i /_\Delta pg_i = \eta_i$，其中 η_i 和 $i = 1, 2$。对规则 MATCH-TRUE 应用逆转规则，可得 $i = 1, 2$

$$v_i /_\Delta p_i = \eta_i$$
$$\mathcal{G}_\Delta(g_i\eta_i) = true$$

根据引理 A.15(ii) 定义域(η_i) =pbv(p_i)。由 pbv(p_1) ∩pbv(p_2) = ∅可得$\eta_1 \sim \eta_2$。因此，根据规则 匹配对，有 $v /_\Delta p = \eta_1 \cup \eta_2$。此外，根据 g_i的构造，我们还有 $G_\Delta(g\eta)$ = $True$，对于 $\eta = \eta_1 \cup \eta_2$成立。因此，根据规则 MATCH-TRUE，有 $v /_\Delta pg = \eta$。

分类讨论结束。

命题 *(ii)* 的证明。假设 $pg = p$ 当 g。逆转规则MATCH-TRUE 可得 $v /_\Delta p = \eta$ 以及$\mathcal{G}_\Delta($ $g\eta)$ = $True$。根据引理A.24和引理 A.25，于是

$$\mathsf{env}(g) \neq \bot$$
$$\mathsf{safe}^{\uparrow}(g) = true$$

于是，$[\![pg]\!]^{\uparrow} = [\![p]\!]^{\uparrow}$ 环境(g)。根据引理 A.26，于是 $\Sigma \cdot \Gamma \vdash v : [\![p]\!]^{\uparrow}$ 环境(g)，如所需。

引理 A.28（模式环境的子类型关系）。令 p 为一个模式，t、t' 为两种类型，且 $t /\!\!/ p$ 和 $t' /\!\!/ p$ 有定义。若 $t \leq t'$，则 $(t /\!\!/ p) \leq (t' /\!\!/ p)$。

证明。对 p进行归纳法。定义 $\Gamma = t /\!\!/ p$ 和 $\Gamma' = t' /\!\!/ p$。

根据 p的结构进行分类讨论。

- 情形 $p = v$ 或 $p =_$或 $p = \$x$：无需证明，因为 定义域$(\Gamma)$ = ∅ =定义域(Γ')。
- 情形 $p = x$：根据定义，定义域(Γ) = $\{x\}$ = 定义域(Γ')，并且根据假设，$\Gamma(x)$ $= t \leq t' = \Gamma'(x)$。
- 情形 $p = (p_1, p_2)$：根据假设，$t \leq t'$，因此由性质 A.2，$\pi_i(t) \leq \pi_i(t')$。应用归纳假设可得

$$(\pi_i(t) /\!\!/ p_i) \leq (\pi_i(t') /\!\!/ p_i)$$

Then

$$t /\!\!/ (p_1, p_2) = (\pi_1(t) /\!\!/ p_1) \between (\pi_2(t) /\!\!/ p_2) \leq (\pi_1(t') /\!\!/ p_1) \between (\pi_2(t') /\!\!/ p_2) = t' /\!\!/ (p_1, p_2)$$

分类讨论结束。

引理 A.29（受保护模式环境的子类型关系）。设 pg 为一个带守卫模式，且 t、t' 为两个类型，使得 $t /\!\!/ p$ 和 $t' /\!\!/ p$ 有定义。若 $t \leq t'$，则 $(t /\!\!/ pg) \leq (t' /\!\!/ pg)$。

证明。设 $pg = p$ 当 g 时。定义 Γ 和 Γ' 如下：

$$\Gamma := t /\!\!/ pg = t /\!\!/ p \between \mathsf{env}(g)$$
$$\Gamma' := t' /\!\!/ pg = t' /\!\!/ p \between \mathsf{env}(g)$$

由引理 A.28 $t /\!\!/ p \leq t' /\!\!/ p'$ 可知。因此 定义域(Γ) = 定义域(Γ')。假设 $x \in$ 定义域(Γ)。若 $x \in$ 定义域$(t /\!\!/ p)$，则 $x \in$ 定义域$(t' /\!\!/ p)$，且 $(t /\!\!/ p)(x) \leq t' /\!\!/ p(x)$。因此，根据环境交集的定义，$\Gamma($ $x) \leq \Gamma'(x)$。

引理 A.30. 假设 $\Sigma \cdot \Gamma \vdash v : t$ 且 $v /_\Delta p = \eta$。若 $x \in$ 定义域(η) 且 $\eta(x)$ #gt，则 $\Sigma \cdot \Gamma \vdash \eta(x) : \mathsf{ty}(gt)$。

证明. 对 p 使用归纳法。
根据 p的形式进行分类讨论。

- 分类讨论 $p = v$，或 $p =$ 或 $p = \$y$：不可能，因为在这种情况下，$v /_\Delta p = \Omega$ 或 $v /_\Delta p = []$。
- 分类讨论 $p = x$：那么 $\eta = [v/x]$，因此 $\eta(x) = v$ 以及 $v \# gt$ 根据假设成立。
  现在由引理 A.13即可得到该结论。
- 分类讨论 $p = (p_1, p_2)$：那么 $v = (v_1, v_2)$，并且

$$v_i /_\Delta p_i = \eta_i \quad (i = 1, 2)$$

$$\eta_1 \sim \eta_2$$

$$\eta = \eta_1 \cup \eta_2$$

由假设 $\Sigma \cdot \Gamma \vdash v : t$，结合引理 A.6 可知存在某个 t_1, t_2

$$\Sigma \cdot \Gamma \vdash v_i : t_i \quad (i = 1, 2)$$

$$t_1 \times t_2 \leq t$$

假设 $x \in$ 定义域(η)，且 $\eta(x) \# gt$。那么 $x \in$ 定义域(η_1) 和/或 $x \in$ 定义域(η_2)，并且如果 x 同时属于两个定义域，那么 $\eta_1(x) = \eta_2(x)$
现在假设 $i \in \{1, 2\}$，且 $x \in$定义域(η_i)。那么 $\eta_i(x) = \eta(x)$，因此 $\eta_i(x) \# gt$。应用归纳假设可得

$$\Sigma \cdot \Gamma \vdash \eta_i(x) : \mathsf{ty}(gt)$$

现在由 $\eta_i(x) = \eta(x)$ 可得所述结论。

分类讨论结束。

定义 *A.31*。对于某个有限映射 F 和某个集合 A，我们用 $\mathrm{F} \restriction_A$ 表示定义域受限为 A 的有限映射；也就是说，$\mathrm{F} \restriction_A = \{x : \mathrm{F}(x) \mid x \in A \cap$ 定义域$(\mathrm{F})\}$。

引理 A.32。
(i) $safe^{\downarrow}(g, A)$ 蕴含 $safe^{\downarrow}(g\eta, A)$，前提是 $dom(\eta) \cap A = \emptyset$。*(ii)* $[\![p]\!]^{\downarrow}_{\Gamma_1} \leq [\![p\eta]\!]^{\downarrow}_{\Gamma_2}$ 前提是 $pbv(p) \cap dom(\eta) = \emptyset$ 且 $\Gamma_1 \restriction_{\mathsf{pbv}(p)} = \Gamma_2 \restriction_{\mathsf{pbv}(p)}$。*(iii)* $[\![pg]\!]^{\downarrow} \leq [\![pg\eta]\!]^{\downarrow}$ 前提是 $pbv(pg) \cap dom(\eta) = \emptyset$。

证明。*(i)* 的证明。我们对 g 进行归纳。若 $g = g_1$ 且 g_2，则由归纳假设可得结论。若 $g =$ 为 True 或 $g =$ 为预言机，结论显然成立。因此，假设 g 是一个类型测试。

- $g =$ 是$_{gt}(v)$，且 $v \# gt$。那么 $v\eta$ 是一个值，并且根据引理 A.19，$v\eta \# gt$。因此，安全$^{\downarrow}(g\eta, A)$。
- $g =$ 是$_{gt}(x)$ 且 $x \in A$。根据假设 定义域$(\eta) \cap A = \emptyset$，我们有 $e\eta = x$，因此安全$^{\downarrow}(g\eta, A)$。
- 否则，安全$^{\downarrow}(g, A) = false$，因此无须证明。

（*ii*）的证明。对 p 进行归纳。假设 $[\![p]\!]^{\downarrow}_{\Gamma_1} \neq \bot$，否则无须证明，因为$\bot \leq t$ 对于任意 t 根据性质 A.3(i)。

- $p = v$。由 $[\![p]\!]^{\downarrow}_{\Gamma_1} \neq \bot$，我们有 $v = \kappa$ 且$\mathsf{ty}(\kappa) = \kappa$ 且 $[\![p]\!]^{\downarrow}_{\Gamma_1} = \kappa$，其中某个 κ。于是 $p\eta = \kappa$ 且 $[\![p\eta]\!]^{\downarrow}_{\Gamma_2} = \kappa$。
- $p =$ 。显然，因为对于任意 Γ，$p\eta =$ $= p$ 且 $[\![p\eta]\!]^{\downarrow}_{\Gamma} = \top$。
- $p = x$。于是 $x \in \mathsf{pbv}(p)$，因此 $p\eta = p = x$ 根据假设 $\mathsf{pbv}(p) \cap$ 定义域$(\eta) = \emptyset$。根据假设 $\Gamma_1 \restriction_{\mathsf{pbv}(p)} = \Gamma_2 \restriction_{\mathsf{pbv}(p)}$，结论得证。

- $p = \$x$。不可能，因为 $[\![p]\!]^{\downarrow}_{\Gamma_1} \neq \bot$。
- $p = (p_1,\ p_2)$。由归纳假设可得。

（*iii*）的证明。假设$pg = p$ 当 g。根据假设$\mathsf{pbv}(pg)$ ∩定义域$(\eta) = \emptyset$，我们还有$\mathsf{pbv}(p)$ ∩ 定义域$(\eta) = \emptyset$。如果 $[\![pg]\!]^{\downarrow} = \bot$，则结论显然成立。否则：

$$[\![pg]\!]^{\downarrow} = [\![p]\!]^{\downarrow}_{\mathsf{env}(g)}$$
$$\mathsf{safe}^{\downarrow}(g, \mathsf{pbv}(p))$$
$$\mathsf{env}(g) \neq \bot$$
$$p \text{ linear}$$

因此，由 (i) 安全$^{\downarrow}(g\eta,\ \mathsf{pbv}(p\eta))$因为 $\mathsf{pbv}(p) = \mathsf{pbv}(p\eta)$。根据引理 A.20环境$(g\eta) \subseteq$环境$(g)$和 环境 $(g\eta)\restriction_{\mathsf{pbv}(p)}$ = 环境 $(g)\restriction_{\mathsf{pbv}(p)}$。因此，环境 $(g\eta) \neq \bot$。此外，$p\eta$ 是线性的，因此

$$[\![pg]\!]^{\downarrow} = [\![p]\!]^{\downarrow}_{\mathsf{env}(g)} \overset{\text{(ii)}}{\leq} [\![p\eta]\!]^{\downarrow}_{\mathsf{env}(g\eta)} = [\![pg\eta]\!]^{\downarrow}$$

引理 A.33。如果安全$^{\uparrow}(g\eta)$ 且环境$(g\eta) \neq \bot$，那么环境$(g) \neq \bot$。

证明。定义 $\Gamma =$ 环境(g)。假设 $\Gamma = \bot$。那么存在 $x \in$ 定义域(Γ)，使得 $\Gamma(x) = \bot$。因此，g 至少包含两个类型测试 $\mathsf{is}_{gt_1}(x)$ 和 $\mathsf{is}_{gt_2}(x)$，使得

$$\mathsf{ty}(gt_1) \wedge \mathsf{ty}(gt_2) = \bot \tag{1}$$

根据 环境$(g\eta) \neq \bot$，我们知道 $x \in$ 定义域(η)。根据 安全$^{\uparrow}(g\eta)$ 可得 $\eta(x)\ \#gt_1$ 且$\eta(x)\ \#gt_2$。但这与（1）矛盾。因此，$\Gamma \neq \bot$。

引理 A.34。

(i) $\mathit{safe}^{\uparrow}(g\eta)$ 蕴含 $\mathit{safe}^{\uparrow}(g)$。*(ii)* $[\![p\eta]\!]^{\uparrow}_{\Gamma_1} \leq [\![p]\!]^{\uparrow}_{\Gamma_2}$在 $\mathit{pbv}(p) \cap \mathit{dom}(\eta) = \emptyset$ 且 $\Gamma_1\restriction_{\mathsf{pbv}(p)} = \Gamma_2\restriction_{\mathsf{pbv}(p)}$ 的条件下。*(iii)* $[\![pg\eta]\!]^{\uparrow} \leq [\![pg]\!]^{\uparrow}$ 在 $\mathit{pbv}(pg) \cap \mathit{dom}(\eta) = \emptyset$ 的条件下。

证明。（*i*）的证明。我们对 g 进行归纳。如果 $g = g_1$ 且 g_2，则由归纳假设可得该命题。如果 $g =$ 为 True 或 $g =$ 为预言机，该命题显然成立。因此，假设 g 是一个类型测试。如果安全$^{\uparrow}(g) = \mathit{True}$，则证明完毕。假设 安全$^{\uparrow}(g) = \mathit{false}$。那么，对于某个值 v（满足 not $v\ \#\ gt$），$g =$ 是$_{gt}(v)$。根据引理 A.19not $v\eta\ \#gt$。因此，安全$^{\uparrow}(g\eta) = \mathit{false}$，这就产生了矛盾。

（*ii*）的证明。对 p 进行归纳。假设 $[\![p\eta]\!]^{\uparrow}_{\Gamma_1} \neq \bot$，否则无需证明。

- $p = v$。如果 v 是某个 κ，那么 $v\eta = \kappa$ 且$[\![v\eta]\!]^{\uparrow}_{\Gamma_1} = \mathsf{ty}(\kappa) = [\![v]\!]^{\uparrow}_{\Gamma_2}$。
- $p =$ ，显然。
- $p = x$。于是 $x \in \mathsf{pbv}(p)$，因此根据假设 $p\eta = p = x\ \mathsf{pbv}(p) \cap$ 定义域$(\eta) = \emptyset$。根据假设 $\Gamma_1\restriction_{\mathsf{pbv}(p)} = \Gamma_2\restriction_{\mathsf{pbv}(p)}$，结论成立。
- $p = \$x$。于是$[\![\$x]\!]^{\uparrow}_{\Gamma_2} = \top$。
- $p = (p_1,\ p_2)$。由归纳假设可得。

（*iii*）的证明。设$pg = p$ 当 g 时。根据假设$\mathsf{pbv}(pg)$ ∩定义域$(\eta) = \emptyset$，我们也有$\mathsf{pbv}(p)$ ∩ 定义域$(\eta) = \emptyset$。如果 $[\![pg\eta]\!]^{\uparrow} = \bot$，则结论显然成立。否则

$$[\![pg\eta]\!]^{\uparrow} = [\![p\eta]\!]^{\uparrow}_{\mathsf{env}(g\eta)}$$
$$\mathsf{safe}^{\uparrow}(g\eta)$$
$$\mathsf{env}(g\eta) \neq \bot$$

根据引理 A.33，于是环境$(g) \neq \bot$。由 (i) 也有安全$^{\uparrow}(g)$。根据引理A.20 环境$(g\eta)\restriction_{\mathsf{pbv}(p)}$ =环境$(g)\restriction_{\mathsf{pbv}(p)}$。因此

$$[\![pg\eta]\!]^{\uparrow} = [\![p\eta]\!]^{\uparrow}_{\mathsf{env}(g\eta)} \overset{\text{(ii)}}{\leq} [\![p]\!]^{\uparrow}_{\mathsf{env}(g)} = [\![pg]\!]^{\uparrow}$$

引理 A.35（模式的环境生成的正确性）。假设$\Sigma \cdot \Gamma \vdash v : t$ 且 $v /_{\Delta} p = \eta$。那么对于所有 $x \in$ 定义域(η)： $\Sigma \cdot \Gamma \vdash x\eta : (t /\!\!/ p)(x)$。

证明。 对 p 进行归纳。

对 p的形状进行分类讨论。

- 情形 $p = v$ 或 $p =$ 或 $p = \$y$：不可能，因为要么是 $\eta = []$，要么是 $v /_{\Delta} p = \Omega$。
- 情形 $p = y$：那么 $\eta = [v/y]$ 且 $t /\!\!/ p = \{y : t\}$。若 $x \in$ 定义域(η)，那么 $x = y$ 且$x\eta = v$ 且 $(t /\!\!/ p)(x) = t$，因此由假设 $\Sigma \cdot \Gamma \vdash v : t$ 可得结论。
- 情形 $p = (p_1, p_2)$：那么

$$t /\!\!/ p = (\pi_1(t) /\!\!/ p_1) \curlywedge (\pi_2(t) /\!\!/ p_2) \tag{1}$$

$$v = (v_1, v_2)$$

$$v_i /\!\!/ p_i = \eta_i \quad (i = 1, 2) \tag{2}$$

$$\eta_1 \sim \eta_2$$

$$\eta = \eta_1 \cup \eta_2$$

根据假设 $\Sigma \cdot \Gamma \vdash v : t$ 以及引理 A.6 对于某个 t_1, t_2

$$\Sigma \cdot \Gamma \vdash v_i : t_i \tag{3}$$

$$t_1 \times t_2 \leq t$$

将归纳假设应用于 (2) 和 (3)，对于 $i = 1, 2$ 得到

$$\Sigma \cdot \Gamma \vdash x\eta_i : (t_i /\!\!/ p_i)(x) \quad (\forall x \in \mathsf{dom}(\eta_i)) \tag{4}$$

现在假设 $x \in$ 定义域$(t /\!\!/ p)$。由 (1)，我们有三种可能的情形。

– $x \in$ 定义域$(\pi_1(t) /\!\!/ p_1)$ 且 $x \notin$ 定义域$(\pi_2(t) /\!\!/ p_2)$ 由 (2) 和引理 A.15，那么 $x \in$ 定义域(η_1) 且 $x \notin$ 定义域(η_2)。

因此， $x\eta = x\eta_1$ 以及 $(t /\!\!/ p)(x) = (\pi_1(t) /\!\!/ p_1)(x)$。根据性质 A.2（iii）同样有$t_1 \leq \pi_1(t)$。因此，结合（4）和引理A.28

$$\Sigma \cdot \Gamma \vdash x\eta_1 : (\pi_1(t) /\!\!/ p_1)(x)$$

如所要求。

– $x \notin$ 定义域$(\pi_1(t) /\!\!/ p_1)$ 和 $x \in$ 定义域$(\pi_2(t) /\!\!/ p_2)$。与前一种情况类似。– $x \in$ 定义域$(\pi_1(t) /\!\!/ p_1)$ 和 $x \in$ 定义域$(\pi_2(t) /\!\!/ p_2)$。于是 $x \in$ 定义域(η_1) 和 $x \in$定义域(η_2)，且 $\eta_1(x) = \eta_2(x)$。此外，

$$(t /\!\!/ p)(x) = (\pi_1(t) /\!\!/ p_1)(x) \wedge (\pi_2(t) /\!\!/ p_2)(x)$$

按照第一种情况中的相同推理，我们可以证明

$$\Sigma \cdot \Gamma \vdash x\eta_i : (\pi_i(t) /\!\!/ p_i)(x)$$

对于 $i = 1, 2$。然后根据引理 A.22

$$\Sigma \cdot \Gamma \vdash x\eta : (t /\!\!/ p)(x)$$

按要求。

结束分类讨论.

定义 *A.36.* 函数环境 Δ 与类型模式环境Σ相符，记作 $\Delta \bowtie \Sigma$，当且仅当满足以下条件：

- Δ = 定义$_{i\in I}$
- 环境(定义 $_i$) $= \Sigma_i$; Σ_i 对所有 $i \in I$
- $\Sigma_{i\in I} \vdash$ 定义$_i$ 良好 对所有 $i \in I$
- $\Sigma = \Sigma_{i\in I}$

引理 A.37（带守卫模式的环境生成的正确性）。假设 $\Sigma \cdot \Gamma \vdash v : t$ 且 $v /_\Delta pg = \eta$ 且 $\Delta \bowtie \Sigma$。那么对于所有 $x \in$ 定义域(η): $\Sigma \cdot \Gamma \vdash x\eta : (t /\!\!/ pg)(x)$。

证明。假设 $pg = p$ 当 g。由 $v /_\Delta pg = \eta$ 可得

$$v /_\Delta p = \eta \tag{1}$$

$$\mathcal{G}_\Delta(g\eta) = true \tag{2}$$

假设 $x \in$ 定义域(η)。根据引理 A.15 可得 $x \in$ 定义域$(t /\!\!/ pg)$。根据定义，我们有 e

$$t /\!\!/ pg = t /\!\!/ p \twoheadrightarrow \mathsf{env}(g) \tag{3}$$

- $x \in$ 定义域$(t /\!\!/ p)$ 且 $x \notin$ 定义域(环境(g))。由引理 A.35即可得出结论。
- $x \notin$ 定义域$(t /\!\!/ p)$ 且 $x \in$ 定义域(环境(g))。根据引理 定义域(η) = 定义域$(t /\!\!/ p)$，这是不可能的，因为引理 A.15。
- $x \in$ 定义域$(t /\!\!/ p)$ 且 $x \in$ 定义域(环境(g))。根据引理A.35 可得

$$\Sigma \cdot \Gamma \vdash x\eta : (t /\!\!/ p)(x) \tag{4}$$

由 $x \in$ 定义域(环境(g))，我们知道 g 包含针对 的类型测试$gt_i(x)$，适用于 $i \in I \neq \emptyset$。由 (2) 我们知道

$$\eta(x) \# gt_i \quad (\forall i \in I)$$

因此，所有 守卫类型$_i$ 都相同。将这个守卫类型称为 守卫类型。根据假设 $\Sigma \cdot \Gamma \vdash v : t$, (1),$x \in$ 定义域 (η)，以及 $\eta(x)$ # 守卫类型，我们可以应用引理 A.30 并得到

$$\Sigma \cdot \Gamma \vdash x\eta : \mathsf{ty}(gt)$$

我们还有 环境$(g)(x)$ = ty(守卫类型)，因此结合 (4) 和 (3)，即可得到 $\Sigma \cdot \Gamma \vdash x\eta : (t /\!\!/ pg)(x)$，如所需。

引理 A.38。假设 $\Sigma \cdot \Gamma \vdash v : t$，其中 t。令 $\Gamma' = u /\!\!/ pg$ 和 $\eta = [v/x]$。若 $x \in$ 定义域(Γ') 且 $x \notin \mathsf{pbv}(pg)$ 以及 $[\![pg\eta]\!]^\uparrow \neq \bot$，则 $\Sigma \cdot \Gamma \vdash v : \Gamma'(x)$。

证明。假设 $pg = p$ 当 g。根据 $x \in$ 定义域(Γ')，我们知道 g 包含类型测试（is$_{gt_i}$ $(x))_{i\in I}$ 对 $I \neq \emptyset$，并且

$$\Gamma'(x) = \bigwedge_{i\in I} \mathsf{ty}(gt_i)$$

根据 $[\![\mathrm{pg}\eta]\!]^\uparrow \neq \bot$ 和 $x \notin \mathrm{pbv(pg)}$，我们知道 $\mathrm{safe}^\uparrow(g\eta)$ = 为 True。因此，对所有$i \in I$, $\mathrm{safe}^\uparrow(\mathrm{is}_{gt_i}(v))$。所以，对所有 $i \in I$, $v \# \mathrm{gt}_i$。由引理 A.13 可得，对所有 $i \in I$, $\Sigma \cdot \Gamma \vdash v : \mathrm{ty(gt}_i)$。然后根据引理 A.22, $\Sigma \cdot \Gamma \vdash v : \Gamma'(x)$, 如所需。

引理 A.39（值替换引理）。假设 $x \notin$ 定义域(Γ)。若 $\Sigma \cdot \Gamma,\{x : t'\} \vdash e : t$且 $\Sigma \cdot \Gamma \vdash v : t'$, 则 $\Sigma \cdot \Gamma \vdash e[v/x] : t$。

证明。 定义 $\Gamma' = \{x : t'\}$ 和 $\eta = [v/x]$。我们对 $\Sigma \cdot \Gamma, \Gamma' \vdash e : t$ 的类型推导进行归纳。

对推导中的最后一条规则进行分类讨论。

- 分类讨论 D-VAR：我们区分两种情况：– $e = x$：那么 $e\eta = v$ 和 $t = t'$，因此 $\Sigma \cdot \Gamma \vdash e\eta$：根据假设，$t$。– $e = y \neq x$：那么 $e\eta = y$ 和 $t = \Gamma(y)$，因此 $\Sigma \cdot \Gamma \vdash e\eta$：根据规则 D-VAR，$t$。
- 分类讨论 D-VAR-POLY：那么 $e = y \neq x$ 和 $t \in \mathsf{inst}(\Sigma(y))$，因此根据规则D-VAR-POLY，$\Sigma \cdot \Gamma \vdash e\eta : t$。
- 分类讨论 D-CONST：显然。
- 分类讨论 D-PAIR：由归纳假设可直接得出。
- 情形 D-APP：由归纳假设可直接得出。
- 情形 D-ABS：于是 $e = \lambda y.\, e'$。不妨设 y 新鲜。根据规则$\Sigma \cdot \Gamma, \Gamma', y$ 的前提：$t_1 \vdash e' : t_2$ 对 $t = t_1 \to t_2$ 成立。于是 $\Sigma \cdot \Gamma, y : t_1 \vdash e\eta$：由归纳假设可得。因此 $\Sigma \cdot \Gamma \vdash e\eta$：通过规则D-ABS得到。
- 情形 D-CASE：于是 $e = \mathsf{case}\ e'\ \mathsf{of}\ (pg_i \to e_i)_{i \in I}$。反演该规则可得：

$$\Sigma \cdot \Gamma, \Gamma' \vdash e' : u \tag{1}$$

$$u \leq \bigvee_{i \in I} [\![pg_i]\!]^{\downarrow} \tag{2}$$

$$t = \bigvee_{i \in I} t'_i \tag{3}$$

并且对所有 $i \in I$:

$$\mathsf{fv}(pg_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Gamma') \tag{4}$$

$$t_i = (u \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow}$$

$$\Gamma_i = (\Gamma, \Gamma') \mathbin{\Rightarrow\!\!\!\!\!\!\Rightarrow} (t_i /\!\!/ pg_i) \tag{5}$$

$$\begin{cases} t'_i = \bot \text{ and } \mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma_i) & \text{if } t_i \leq \bot \\ \Sigma \cdot \Gamma_i \vdash e_i : t'_i & \text{otherwise} \end{cases} \tag{6}$$

不妨设 $\mathsf{pbv}(pg_{i \in I})$ 新鲜，于是

$$\mathsf{pbv}(pg_i) \cap \mathsf{dom}(\eta) = \emptyset \quad (\forall i \in I) \tag{7}$$

$$e\eta = \mathsf{case}\ e'\eta\ \mathtt{of}\ (pg_i\eta \to e_i\eta)_{i \in I}$$

对（1）应用归纳假设可得

$$\Sigma \cdot \Gamma \vdash e'\eta : u \tag{8}$$

根据（7）和引理 A.32，对于任意 $i \in I$，有 $[\![pg_i]\!]^{\downarrow} \leq [\![pg_i\eta]\!]^{\downarrow}$。根据（2），于是

$$u \leq \bigvee_{i \in I} [\![(pg_i\eta)]\!]^{\downarrow} \tag{9}$$

对所有 $i \in I$定义

$$t^{\star}_i = (u \setminus \bigvee_{j<i} [\![pg_j\eta]\!]^{\downarrow}) \wedge [\![pg_i\eta]\!]^{\uparrow} \tag{10}$$

$$\Gamma^{\star}_i = t^{\star}_i /\!\!/ pg_i\eta \tag{11}$$

根据（7）和引理A.32 以及引理A.34，对于任意 $i \in I$，有 $[\![pg_i]\!]^{\downarrow} \trianglelefteq [\![pg_i\eta]\!]^{\downarrow}$ 且 $[\![pg_i\eta]\!]^{\uparrow} \leq [\![pg_i]\!]^{\uparrow}$。对于任意 $i \in I$，由集合论推理可得

以及 $t_i^\star$和 t_i 的定义

$$t_i^\star \leq t_i \tag{12}$$

由（4）及 η 的定义可得

$$\mathsf{fv}(pg_i\eta) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma) \tag{13}$$

现在假设 $i \in I$。

– 情形 $t_i^\star \leq \bot$。由（6）可得（在 $t_i \not\leq \bot$ 的情形下使用引理 A.17 ）$\mathsf{fv}(e_i) \subseteq$ 定义域(Σ) ∪定义域(Γ_i)。因此

$$\mathsf{fv}(e_i\eta) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma_i^\star) \tag{14}$$

在此情形下定义 $t_i^{\star\prime} = \bot$ 。

– 情形 $t_i^\star \not\leq \bot$。定义

$$\widetilde{\Gamma}_i = t_i /\!\!/ pg_i$$

$$\widetilde{\widetilde{\Gamma}}_i = t_i /\!\!/ (pg_i\eta)$$

$$t_{i,x} = \begin{cases} \widetilde{\Gamma}_i(x) & \text{if } x \in \mathsf{dom}(\widetilde{\Gamma}_i) \\ \top & \text{otherwise} \end{cases}$$

~ ~ ~若 $x \in$ 定义域(Γ_i)，根据引理 A.21 $\Gamma_i = \Gamma_i$， x ，可得：$t_{i,\ x}$。否则，同一~ ~引理可得 $\Gamma_i = \Gamma_i$。因此，结合（5）和t_i的x 定义，我们有

$$\Gamma_i = (\Gamma, \Gamma') \between \widetilde{\Gamma}_i = (\Gamma, \{x : t'\}) \between \widetilde{\Gamma}_i = (\Gamma \between \widetilde{\widetilde{\Gamma}}_i), x : (t' \wedge t_{i,x}) \tag{15}$$

根据假设 $\Sigma \cdot \Gamma \vdash v :\ \ t'$，结合引理A.8 和引理 A.9

$$\Sigma \cdot \Gamma \between \widetilde{\widetilde{\Gamma}}_i \vdash v : t' \tag{16}$$

结合 $t_i^\star \not\leq \bot$ 和（10），我们得到 $[\![\mathrm{pg}_i\eta]\!]^\uparrow \neq \bot$。因此，若 $x \in \mathsf{dom}(\widetilde{\Gamma}_i)$，根据 ~ ~引理 A.38，可得 $\Sigma \cdot \Gamma \vdash v : t_{i,x}$，其中 $t_{i,x} = \Gamma_i(x)$。否则 $x \notin \mathsf{dom}(\Gamma_i)$，因此我们有 $\Sigma \cdot \Gamma \vdash v : t_{i,x}$，因为此时 $t_{i,x} = \top$。结合（16）、引理 A.8 和引理 A.9，

$$\Sigma \cdot \Gamma \between \widetilde{\widetilde{\Gamma}}_i \vdash v : (t' \wedge t_{i,x}) \tag{17}$$

由 $t_i^\star \not\leq \bot$ 和（12）可得 $t_i \not\leq \bot$。然后由（6）和（15）

$$\Sigma \cdot (\Gamma \between \widetilde{\widetilde{\Gamma}}_i), x : (t' \wedge t_{i,x}) \vdash e_i : t_i' \tag{18}$$

将归纳假设应用于（17）和（18）可得

$$\Sigma \cdot \Gamma \between \widetilde{\widetilde{\Gamma}}_i \vdash e_i\eta : t_i'$$

~ ~ ~

由~（11），根据 Γ_i 的定义，结合（12）和引理 A.29，可得 $\Gamma_i^\star \leq \Gamma_i$，因此根据引理 A.8

$$\Sigma \cdot \Gamma \between \Gamma_i^\star \vdash e_i\eta : t_i' \tag{19}$$

定义 $t_i^{\star\prime} = t_i'$。至此， $t_i \not\leq \bot$ 的情况处理完毕。

将 (8)、(9)、(13)、(10)、(14) 和 (19) 作为规则 D-CASE 的前提，可得

$$\Sigma \cdot \Gamma \vdash e\eta : \bigvee_{i \in I} t_i^{\star\prime}$$

根据 $t_i^{\star\prime}$的构造，我们有 $t_i^{\star\prime} \leq t_i'$对所有 $i \in I$均成立。因此，$\bigvee_{i \in I} t_i^{\star\prime} \leq \bigvee_{i \in I} t_i'$。使用规则 D-SUB 以及（3）即可得到 $\Sigma \cdot \Gamma \vdash e\eta : t$，证毕。

- 分类讨论 D-SUB：由归纳假设可直接得出。

分类讨论结束。

引理 *A.40*（多值替换引理）。假设 $\Sigma \cdot \Gamma, \Gamma' \vdash e : t$，且 $dom(\Gamma) \cap dom(\Gamma') = \emptyset$。设 η 为一个值替换，其 $dom(\eta) = dom(\Gamma')$，并且$\Sigma \cdot \Gamma \vdash \eta(x) : \Gamma'(x)$ 对所有 $x \in dom(\eta)$ 均成立。那么 $\Sigma \cdot \Gamma \vdash e\eta : t$。

证明。对 Γ'中的元素个数进行归纳，并使用引理 A.39。

引理 A.41。若 $\Sigma \cdot \Gamma \vdash v : \bigvee_{i \in I} [\![pg_i]\!]^{\downarrow}$ 且 $I \neq \emptyset$，则存在一个 $j \in I$，使得$\Sigma \cdot \Gamma \vdash v : [\![pg_j]\!]^{\downarrow}$。

证明。定义如下简单类型文法：

$$s ::= \top \mid \bot \mid \iota \mid \texttt{int} \mid \texttt{float} \mid \bot \to \top \mid s \times s$$

不难验证，对于任意守卫模式 pg，都存在某个简单类型 s，使得 $[\![pg]\!]^{\downarrow}$ 等价于 s。（对于任意守卫 g，环境(g) 将变量映射到简单类型。对于任意模式 p，$[\![p]\!]^{\downarrow}_{\Gamma}$ 是一个简单类型，因为 Γ 只包含简单类型。）

证明的其余部分依赖于类型模型，参见 Castagna 等人 [13, 引理 A.14，第 27] 页及 [10, 22]。

定理 A.42（表达式进展性）。假设 $\Sigma \cdot \emptyset \vdash e : t$ 且满足 $\Delta \bowtie \Sigma$。那么，e 要么是一个值，要么存在 $e^{\star}$ 使得 $\Delta \vdash e \rightsquigarrow e^{\star}$。

证明。对 $\Sigma \cdot \emptyset \vdash e : t$ 的推导进行归纳。分类讨论推导的最后一条规则。

- 情形 D-VAR：则 $e = x$，且 $x \in$ 定义域$(\emptyset)$，这是不可能的。
- 情形 D-VAR-POLY：则 $e = x$，其中某个 $x \in$ 定义域(Σ)。由 $\Delta \bowtie \Sigma$，我们知道 $(x = v) \in \Delta$ 或 $(x : \sigma) = v \in \Delta$。因此，$x$ 可根据 RED-VAR 和 RED-CONTEXT归约为 v。
- 情形 D-CONST、D-ABS：e 是一个值。
- 情形 D-APP：则 $e = e_1\, e_2$，且 $\Sigma \cdot \emptyset \vdash e_1 : t_\prime \to t$ 和 $\Sigma \cdot \emptyset \vdash e_2 : t_\prime$。根据归纳假设，我们知道以下情况之一成立：– e_1 执行一步归约。因此 e 也执行一步归约。– e_1 是一个值，且 e_2 执行一步归约。因此 e 也执行一步归约。– e_1 和 e_2 都是值。根据引理 A.10，我们知道 e_1 是一个 λ 抽象，因此 e可根据 RED-ABS 和 RED-CONTEXT执行一步归约。
- 情形 D-PAIR：由归纳假设可直接得出。
- 情形 D-CASE：于是 $e = \text{case } e_\prime \text{ of } (pg_i \to e_i)_{i \in I}$。反演该规则可得：

$$\Sigma \cdot \emptyset \vdash e' : t' \tag{1}$$

$$t' \leq \bigvee_{i \in I} [\![pg_i]\!]^{\downarrow} \tag{2}$$

$$t = \bigvee_{i \in I} t_i'$$

并且对于所有 $i \in I$:

$$\mathsf{fv}(pg_i) \subseteq \mathsf{dom}(\Sigma) \tag{3}$$

$$t_i = (t' \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow}$$

$$\Gamma_i = t_i /\!\!/ pg_i$$

$$\begin{cases} t'_i = \bot \text{ and } \mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma_i) & \text{if } t_i \leq \bot \\ \Sigma \cdot \Gamma_i \vdash e_i : t'_i & \text{otherwise} \end{cases}$$

将归纳假设应用于（1）可知， e' 要么归约，要么是一个值。如果 e' 归约，那么根据规则 RED-CONTEXT， e 也会归约，我们就完成了。
因此，假设 e' 是一个值， v。根据（1）、（2）以及引理A.41，我们知道存在 $j \in I$ 使得

$$\Sigma \cdot \emptyset \vdash v : [\![pg_j]\!]^{\downarrow}$$

根据引理A.27(i)，对于某个 η' ，我们有 $v /_{\Delta} pg_j = \eta'$。因此，存在$k \leq j$满足

$$v /_{\Delta} pg_k = \eta \quad \text{for some } \eta$$

$$v /_{\Delta} pg_{k'} = \Omega \quad (\forall k' < k)$$

根据规则 RED-CASE和 RED-CONTEXT，对于 $e^{\star} = e_k\eta$ ，我们有 $\Delta \vdash e \quad e^{\star}$。

- 情形 D-SUB：由归纳假设可得。

结束分类讨论。

定理 A.43（程序的进展性）。假设 $\vdash \texttt{letrec}\ \Delta\ \texttt{in}\ e : t$ 对某个 t成立。那么，e 要么是一个值，要么存在一个表达式 $e^{\star}$ ，使得 $\vdash \texttt{letrec}\ \Delta\ \texttt{in}\ e\texttt{letrec}\ \Delta\ \texttt{in}\ e^{\star}$。

证明。由 $\vdash \texttt{letrec}\ \Delta\texttt{in}\ e : t$ 通过逆转规则 d-prog

$$\Delta = def_{i \in I}$$

$$\mathsf{envs}(def_i) = \langle \Sigma_i ; \Sigma'_i \rangle \quad (\forall i \in I)$$

$$\Sigma_{i \in I} \vdash def_i\ \mathsf{ok} \quad (\forall i \in I)$$

$$\Sigma'_{i \in I} \cdot \emptyset \vdash e : t$$

Thus， $\Delta \bowtie \Sigma'_{i \in I}$。由定理 A.42 和归约规则 RED-PROG即可得出结论。

引理 A.44（类型替换引理）。若 $\Sigma \cdot \Gamma \vdash e : t$，则 $\Sigma\theta \cdot \Gamma\theta \vdash e : t\theta$。

证明。
分类讨论关于 $\Sigma \cdot \Gamma \vdash e : t$ 的推导。

- 分类讨论 D-VAR：我们有 $\Sigma \cdot \Gamma \vdash x : t$ ，以及 $\Gamma(x) = t$。根据定义， $\Gamma\theta(x) = t\theta$ ，因此结论成立。
- 分类讨论 d-var-poly：我们有 $\Sigma \cdot \Gamma \vdash x : t, \Sigma(x) = \forall A$ 。 u，且 $t \in \mathsf{inst}(\Sigma(x))$，因此对于某个 $\theta\bigstar$，有 $t = u\theta\bigstar$，并且 $\mathsf{dom}(\theta\bigstar) = A$。通过 α-重命名，我们假设 $A \cap \mathsf{dom}(\theta) = \emptyset$，且 $A \cap \mathsf{ftv}(\theta) = \emptyset$。然后，$(\Sigma\theta)(x) = \forall A . u\theta$。对于 $\theta' = [\alpha\theta\bigstar\theta / \alpha \mid \alpha \in A]$，我们有 $t\theta = u\theta\theta'$，因此 $t\theta \in \mathsf{inst}(\forall A . u\theta)$。应用 d-var-poly 即得结论。

- 情况 D-CONST：这种情况很直接。

- 情况 D-ABS：我们有 $\Sigma\cdot\Gamma\vdash\lambda x.e\prime:\ t_1\to t_2$ 其中 $\Sigma\cdot\Gamma,\ x:\ t_1\vdash e\prime:\ t_2$ 且 $t=t_1\to t_2$。根据归纳假设，我们有 $\Sigma\theta\cdot\Gamma\theta,\ x:\ t_1\theta\vdash e:\ t_2\theta$。然后根据 D-ABS，$\Sigma\theta\cdot\Gamma\theta\vdash\lambda x.e:\ t_1\theta\to t_2\theta$。
- 情形 D-APP 和 D-PAIR：根据归纳假设，结论可直接得出。
- 情形 D-CASE：$e=\mathsf{case}\ e\prime\ \mathsf{of}\ (pg_i\to e_i)_{i\in I}$。反演该规则可得

$$\Sigma\cdot\Gamma\vdash e':u \tag{1}$$

$$u\le\vee_{i\in I}[\![pg_i]\!]^{\downarrow} \tag{2}$$

并且对于所有 $i\in I$：

$$\mathsf{fv}(pg_i)\subseteq\mathsf{dom}(\Sigma)\cup\mathsf{dom}(\Gamma) \tag{3}$$

$$t_i=(u\setminus\vee_{j<i}[\![pg_j]\!]^{\downarrow})\wedge[\![pg_i]\!]^{\uparrow}$$

$$\Gamma_i=\Gamma\mathbin{\mathbb{A}}(t_i/\!\!/pg_i)$$

$$\begin{cases}t_i'=\bot\text{ and }\mathsf{fv}(e_i)\subseteq\mathsf{dom}(\Sigma)\cup\mathsf{dom}(\Gamma_i) & \text{if } t_i\le\bot\\ \Sigma\cdot\Gamma_i\vdash e_i:t_i' & \text{otherwise}\end{cases} \tag{4}$$

$$t=\vee_{i\in I}t_i'$$

由（1）和归纳假设可得：

$$\Sigma\theta\cdot\Gamma\theta\vdash e':u\theta$$

由（2）、性质A.3和引理A.16可得：

$$u\theta\le\vee_{i\in I}[\![pg_i]\!]^{\downarrow}$$

现在假设 $i\in I$。由（3）可得

$$\mathsf{fv}(pg_i)\subseteq\mathsf{dom}(\Sigma\theta)\cup\mathsf{dom}(\Gamma\theta)$$

定义

$$t_i^{\star}=(u\theta\setminus\vee_{j<i}[\![pg_j]\!]^{\downarrow})\wedge[\![pg_i]\!]^{\uparrow}\overset{\text{Lemma A.16}}{=}t_i\theta \tag{5}$$

$$\Gamma_i^{\star}=\Gamma\theta\mathbin{\mathbb{A}}(t_i^{\star}/\!\!/pg_i) \tag{6}$$

– 情形 $t_i^{\star}\le\bot$。定义 $u_i=\bot$。由（4）、引理 A.17 以及 $\mathsf{dom}(\Gamma_i)=\mathsf{dom}(\Gamma_i^{\star})$，我们还有 $\mathsf{fv}(e_i)\subseteq\mathsf{dom}(\Sigma\theta)\cup\mathsf{dom}(\Gamma_i^{\star})$。– 否则，我们必有 $t_i\not\le\bot$，因为 $t_i\le\bot$ 蕴含 $t_i\theta\le\bot$（性质 A.3）。因此，将归纳假设应用于（4）可得

$$\Sigma\theta\cdot\Gamma\theta\vdash e_i:t_i'\theta$$

定义 $u_i:=t_i'\theta$。由（5）和（6），我们还有 $\Gamma_i^{\star}=\Gamma_i\theta$

我们刚刚已经证明了规则 D-CASE 的前提成立。应用该规则可得

$$\Sigma\theta\cdot\Gamma\theta\vdash e:\bigvee_{i\in I}u_i$$

我们有 $u_i\le t_i'\theta$ 对所有 $i\in I$ 都成立。（若 $t_i^{\star}\le\bot$，则 $u_i=\bot\le t_i'\theta$。否则 $u_i=t_i'\theta$。）因此

$$\bigvee_{i\in I}u_i\le\bigvee_{i\in I}t_i'\theta=t\theta$$

因此，按照规则 D-SUB $\Sigma\theta\cdot\Gamma\theta\vdash e:t\theta$，即可满足要求。

- 分类讨论 D-SUB：该结论可直接由归纳假设和性质 A.3(iv) 得出。分类讨论结束。

定理 A.45（表达式的主体归约）。假设 $\Sigma \cdot \Gamma \vdash e : t$ 且定义域(Σ) ∩定义域(Γ) $= \emptyset$ 且 $\Delta \bowtie \Sigma$。若 $\Delta \vdash e \quad e^\star$，则 $\Sigma \cdot \Gamma' \vdash e^\star : t$ 存在某个 Γ' 使得 $\Gamma' \leq \Gamma$。

证明。对 $\Sigma \cdot \Gamma \vdash e : t$ 的推导进行归纳。
分类讨论 推导的最后一条规则。

- 情形 D-VAR：于是 $x \notin$ 定义域(Σ)，因此 $x \notin$ 定义域(Δ)，这是由 $\Delta \bowtie \Sigma$ 得出的。但这样一来 $e = x$ 就无法归约。
- 情形 D-VAR-POLY：于是 $e = x \in$ 定义域(Σ) 且 $t \in \mathsf{inst}(\Sigma(x))$。由 $\Delta \vdash x \hookrightarrow e\star$，通过逆转规则 RED-VAR 可得，要么 $(x = v) \in \Delta$，要么 $(x : \sigma = v) \in \Delta$，且 $e \star = v$。$-(x = v) \in \Delta$。于是根据 $\Delta \bowtie \Sigma$，存在某个单态类型 u 使得

$$\Sigma \cdot \emptyset \vdash v : u$$
$$\Sigma(x) = \forall A.u \quad \text{with } A = \mathsf{ftv}(u)$$

因此，$t = u\theta$ 对于某个 θ 满足 定义域(θ) $= A$。根据引理 A.44 $\Sigma \cdot \emptyset \vdash v : t$（注意ftv(Σ) $= \emptyset$ 这是由 $\Delta \bowtie \Sigma$ 得出的）。根据引理A.9，于是 $\Sigma \cdot \Gamma \vdash v : t$。
$-(x : \sigma = v) \in \Delta$. 然后根据 $\Delta \bowtie \Sigma$，存在单态类型 $t_{i \in I}$ 使得

$$\Sigma(x) = \forall A. \bigwedge_{i \in I} t_i \quad \text{with } A \subseteq \bigvee_{i \in I} \mathsf{ftv}(t_i)$$
$$\Sigma \cdot \emptyset \vdash v : t_i \quad (\forall i \in I)$$

因此，$t = \wedge_{i \in I}(t_i\theta)$ 对于某个 θ，满足 定义域(θ) $= A$。根据引理 A.44 $\Sigma \cdot \emptyset \vdash v : t_i\theta$对所有 $i \in I$。根据引理 A.9 和引理 A.22，于是 $\Sigma \cdot \Gamma \vdash v : t$。

- 情形 D-CONST,D-ABS：这些情形不会出现，因为这些表达式不会规约。
- 情形 D-APP：因此，$e = e_1\, e_2$ 满足 $\Sigma \cdot \Gamma \vdash e_1 : t\prime \to t$，且 $\Sigma \cdot \Gamma \vdash e_2 : t\prime$。分三种情形：

(1) $\Delta \vdash e_1\, e_2 \rightsquigarrow e_1'\, e_2$ 满足 $\Delta \vdash e_1 \rightsquigarrow e_1'$。根据归纳假设，$\Sigma \cdot \Gamma' \vdash e_1' : t' \to t$ 对于 $\Gamma' \leq \Gamma$成立，我们可以结合引理 A.8 再次应用 d-app。(2) $\Delta \vdash e_1\, e_2 \quad e_1\, e_2'$，且 e_1 是一个值，并且 $\Delta \vdash e_2 \quad e_2'$。根据归纳假设，$\Sigma \cdot \Gamma' \vdash e_2' : t'$对于 $\Gamma' \leq \Gamma$ 成立，我们可以结合引理 A.8 再次应用 d-app。(3) $e_1 = \lambda x.e'$，且 $e_2 = v_2$ 对于某些 x, e', v_2 成立。此外，$\Delta \vdash (\lambda x.e')\ v_2 \quad e'[v_2/x]$满足 $e^\star = e'[v_2/x]$。根据假设 $\Sigma \cdot \Gamma \vdash \lambda x.e' : t' \to t$以及引理 A.10，可得 $\Sigma \cdot \Gamma, x : t' \vdash e' : t$。根据假设 $\Sigma \cdot \Gamma \vdash v_2 : t'$ 以及引理 A.39，可得所需的 $\Sigma \cdot \Gamma \vdash e^\star : t$。

- 情形 D-PAIR：$e = (e_1, e_2)$，其中 $\Sigma \cdot \Gamma \vdash e_i : t_i$ 对于 $i = 1, 2$ 和 $t = t_1 \times t_2$。共有三种情形：

(1) $\Delta \vdash (e_1, e_2)\ (e_1', e_2)$，其中 $\Delta \vdash e_1 \quad e_1'$根据归纳假设 $\Sigma \cdot \Gamma' \vdash e_1' : t_1$ 对于 $\Gamma' \leq \Gamma$，我们可以结合引理 A.8 再次应用 d-pair。(2) $\Delta \vdash (v_1, e_2)\ (v_1, e_2')$，其中 $\Delta \vdash e_2 \quad e_2'$。根据归纳假设 $\Sigma \cdot \Gamma' \vdash e_2' : t_2$ 对于 $\Gamma' \leq \Gamma$，我们可以再次应用 d-pair。结合引理 A.8。(3) $e = (v_1, v_2)$。不可能，因为值不会归约。

- 情况 d-case：$e =$ 情况 $e\prime$ 关于 $(pg_i \to e_i)_{i \in I}$ 通过逆转规则 d-case 得到 d-case

$$\Sigma \cdot \Gamma \vdash e' : t' \tag{1}$$
$$t' \leq \bigvee_{i \in I} [\![pg_i]\!]^{\downarrow} \tag{2}$$

并且对于所有 $i \in I$:

$$\mathsf{fv}(pg_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma) \tag{3}$$

$$t_i = (t' \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow} \tag{4}$$

$$\Gamma_i = \Gamma \twoheadleftarrow (t_i /\!\!/ pg_i) \tag{5}$$

$$\begin{cases} t_i' = \bot \text{ and } \mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma_i) & \text{if } t_i \leq \bot \\ \Sigma \cdot \Gamma_i \vdash e_i : t_i' & \text{otherwise} \end{cases} \tag{6}$$

$$t = \bigvee_{i \in I} t_i' \tag{7}$$

对于 $\Delta \vdash e \rightsquigarrow e^{\star}$，我们必须区分两种情况。

(1) $\Delta \vdash e' \rightsquigarrow e''$ 和 $e^{\star} =$ 情形 e'' of $(\mathrm{pg}_i \to e_i)_{i \in I}$ 结合 (1) 和归纳假设，得到$\Sigma \cdot \Gamma' \vdash e'' : t'$ 对于 $\Gamma' \leq \Gamma$。定义 $\Gamma_i' = \Gamma' \twoheadleftarrow (t_i /\!\!/ \mathrm{pg}_i)$，于是 $\Gamma_i' \leq \Gamma_i$。– 若 $t_i \leq \bot$，定义 $t_i' = \bot$，并由 (6) 得到 $\mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma_i')$。– 否则，结合 (6) 和引理 A.8 $\Sigma \cdot \Gamma_i' \vdash e_i : t_i'$

然后结合 D-CASE $\Sigma \cdot \Gamma' \vdash e^{\star} : t$，即可满足要求。

(2) e' 是一个值 v。于是，根据归约规则 RED-CASE 的逆向应用，我们得到存在某个 $j \in I$ 和某个 η，使得

$$v /_{\Delta} pg_j = \eta \tag{8}$$

$$v /_{\Delta} pg_i = \Omega \quad (\forall i < j) \tag{9}$$

$$e^{\star} = e_j \eta$$

由（8），根据引理 A.27（ii）可得；由（9），根据引理 A.27（iii）可得

$$\Sigma \cdot \Gamma \vdash v : [\![pg_j]\!]^{\uparrow} \tag{10}$$

$$\Sigma \cdot \Gamma \vdash v : \neg [\![pg_i]\!]^{\downarrow} \quad (\forall i < j)$$

结合 $e' = v$ 和（1）、（2）、（4），则有

$$\Sigma \cdot \Gamma \vdash v : t_j \tag{11}$$

根据引理 A.12，我们知道 $t_j \not\leq \bot$。因此，由（6）

$$\Sigma \cdot \Gamma_j \vdash e_j : t_j' \tag{12}$$

由（11）、（8）、假设 $\Delta \bowtie \Sigma$ 和引理A.37，我们得到对所有$x \in$定义域(η):

$$\Sigma \cdot \Gamma \vdash \eta(x) : (t_j /\!\!/ pg_j)(x) \tag{13}$$

现在我们的目标是为 $e^{\star} = e_j \eta$ 赋予一个类型。为此，我们定义

$$\widetilde{\Gamma}_j = \Gamma \twoheadleftarrow \{x : (t_j /\!\!/ pg_j)(x) \mid x \in \mathsf{dom}(t_j /\!\!/ pg_j), x \notin \mathsf{pbv}(p_j), x \notin \mathsf{dom}(\Sigma)\}$$

$$\Gamma_j^{\star} = \{x : (t_j /\!\!/ pg_j)(x) \mid x \in \mathsf{dom}(t_j /\!\!/ pg_j), x \in \mathsf{pbv}(p_j)\}$$

不失一般性，

$$\mathsf{dom}(\widetilde{\Gamma}_j) \cap \mathsf{dom}(\Gamma_j^{\star}) = \emptyset \tag{14}$$

~根据定义，我们有 Γ_j 满足 $\Gamma_j = \Gamma_j,\ \Gamma_j^\star$（注意，定义域($\Gamma$) ∩定义域($\Sigma$) = $\emptyset$由假设可知）。因此根据（12）可得

$$\Sigma \cdot \widetilde{\Gamma}_j, \Gamma_j^\star \vdash e_j : t_j' \tag{15}$$

根据引理 A.15

$$\mathsf{dom}(\eta) = \mathsf{pbv}(p_j) = \mathsf{dom}(\Gamma_j^\star) \tag{16}$$

因此，根据（13）$\Sigma \cdot \Gamma \vdash \eta$（$x$）：$\Gamma_j^\star$（$x$）对所有 $x \in$ 定义域（η）。我们稍后证明~$\Gamma_j \leq \Gamma$。然后根据引理A.8

$$\Sigma \cdot \widetilde{\Gamma}_j \vdash \eta(x) : \Gamma_j^\star(x) \quad (\forall x \in \mathsf{dom}(\eta)) \tag{17}$$

将引理 A.40 应用于（14）、（15）、（16）和（17）可得

$$\Sigma \cdot \widetilde{\Gamma}_j \vdash e_j\eta : t_j'$$

根据（7）和 $e^\star = e_j\eta$，再应用规则 D-SUB，则

$$\Sigma \cdot \widetilde{\Gamma}_j \vdash e^\star : t$$

~使用 Γ_j 作为定理中提到的 Γ'，即可完成此情况。

~~~我们仍需证明 $\Gamma_j \leq \Gamma$。根据以下推理可得 定义域(Γ_j) = 定义域(Γ)：

~– dom(Γ_j) ⊆ dom(Γ)。假设 $x \in \mathsf{dom}(\Gamma_j)$。有趣的情形是$x \in \mathsf{dom}(t_j /\!\!/ \mathsf{pg}\ j)$ 且 $x \notin \mathsf{pbv}(p_j)$。于是 $x \in \mathsf{fv}(\mathsf{pg}\ j)$，因此根据（3）和 Γ_j 的定义，$x \in \mathsf{dom}(\Gamma)$~。~~~~~– dom($\Gamma$) ⊆dom($\Gamma_j$)。根据 Γ_j 的构造，这成立。~~现在假设 $x \in \mathsf{dom}(\Gamma_j)$。若 $x \notin \mathsf{dom}(t_j /\!\!/ \mathsf{pg}\ j)$，则显然 $\Gamma_j(x) = \Gamma(x)$。~否则，$\Gamma_j(x) = \Gamma(x) \wedge (t_j /\!\!/ \mathsf{pg}\ j)(x) \leq \Gamma(x)$。

- 分类 D-SUB：由归纳假设和规则 D-SUB可得。

分类讨论结束。

定理 A.46（程序主体归约）。若 $\vdash$ 程序： t 且 $\vdash$ 程序 程序′，则 $\vdash$ 程序↬： t。

证明。 假设程序 = letrec Δin e。那么，根据逆转规则RED-PROG，有程序′ = letrec Δ in e' and $\Delta \vdash e \rightsquigarrow e'$。由 $\vdash$ 程序： t，根据逆转规则D-PROG可得：

$$\Delta = def_{i \in I}$$
$$\mathsf{envs}(def_i) = \langle \Sigma_i; \Sigma_i' \rangle \quad (\forall i \in I)$$
$$\Sigma_{i \in I} \vdash def_i\ \mathsf{ok} \quad (\forall i \in I)$$
$$\Sigma_{i \in I}' \cdot \emptyset \vdash e : t$$

Ther此前，$\Delta \bowtie \Sigma_{i \in I}'$。由定理A.45，则 $\Sigma_{i \in I}' \cdot \emptyset \vdash e'$： t。根据D-PROG，结论成立。

推论 A.47（类型可靠性）。设程序 $prog$ = letrec Δ in e 满足 $\vdash$ 程序： t，其中 t。那么，$prog$ 要么发散，要么归约为程序′ = letrec Δ in v，其中某个值 v满足 $\vdash$ 程序′： t。

证明。 该论断可直接由定理A.43 和定理 A.46得出。
~~~

B 算法式系统的可靠性与完备性

本附录给出相关证明，以说明通过约束生成（图 11）、约束重写（图 12）和约束求解（tally）进行类型重建的算法式系统，相对于声明式类型系统（图10）是可靠且完备的。

图 14 和图 15 包含正文中图 11和图 12 中的约束生成与重写判断的替代定义。这些替代定义会显式跟踪类型变量集 A中的新鲜类型变量。除此之外，替代定义与正文中的定义相同。我们采用如下约定：若未写出类型变量集，则表示存在这样的集合，但不显式命名它。例如，记号 $e : t \Rightarrow C$ 意味着存在某个类型变量集 A，使得 $e : t \Rightarrow_A C$。

B.1 可靠性

我们首先证明，如果一个程序在算法式系统中是可定型的，那么它在声明式系统中也可定型。

引理 B.1（模式的环境生成的可靠性）。给定某个类型 t和模式 p。假设 $t /\!\!/ p \Rightarrow \langle C; \Gamma' \rangle$ 且 $\Sigma \cdot \Gamma \vdash C \quad d$，并且对某些 C、Σ、Γ、Γ'、$\rightsquigarrow_d$ 和 θ成立。定义 $\Gamma^\star = t\theta /\!\!/ p$。则 $\Gamma\theta \curlywedge \Gamma^\star \leq (\Gamma \curlywedge \Gamma')\theta$。

证明。 对 p 的结构进行归纳法。

分类讨论 p 的结构。

- 情形 $p = \_$ 或 $p = v$ 或 $p = \$x$：则 $C = \mathsf{True}$，$\Gamma' = \emptyset$，$t\theta /\!\!/ p = \emptyset = \Gamma_\star$。该命题立即得证
- 情形 $p = y$ 对于某个变量 y：则 $C = \mathsf{True}$ 且 $\Gamma' = \{y : t\}$。进一步 $t\theta /\!\!/ p = \{y : t\theta\} = \Gamma_\star$。显然，

$$\mathsf{dom}(\Gamma \curlywedge \Gamma') = \mathsf{dom}(\Gamma\theta \curlywedge \Gamma^\star)$$

假设 $x \in$ 定义域$(\Gamma \curlywedge \Gamma')$。我们需要证明 $(\Gamma\theta \curlywedge \Gamma^\star)(x) \leq (\Gamma \curlywedge \Gamma')\theta(x)$– 若 $x \neq y$，则有 $(\Gamma\theta \curlywedge \Gamma^\star)(x) = \Gamma(x)\theta = (\Gamma \curlywedge \Gamma')\theta(x)$。– 若 $x = y$ 定义

$$u = \begin{cases} \Gamma(y) & \text{if } y \in \mathsf{dom}(\Gamma) \\ \top & \text{otherwise} \end{cases}$$

Then $(\Gamma\theta \curlywedge \Gamma^\star)(x) = u\theta \wedge t\theta = (u \wedge t)\theta = (\Gamma \curlywedge \Gamma')\theta(x)$.

- 情形 $p = (p_1, \ p_2)$：我们有

$$t /\!\!/ p \Rightarrow \langle C_1 \ \& \ C_2 \ \& \ t \doteq \alpha_1 \times \alpha_2; \Gamma' \rangle$$
$$\Gamma' = \Gamma_1 \curlywedge \Gamma_2$$
$$\alpha_i \text{ fresh} \quad \alpha_i /\!\!/ p_i \Rightarrow \langle C_i; \Gamma_i \rangle \quad \Sigma \cdot \Gamma \vdash C_i \rightsquigarrow d_i \quad \theta \Vdash d_i \quad (i = 1, 2)$$
$$C = C_1 \ \& \ C_2 \quad d = d_1 \ \& \ d_2$$
$$\theta \Vdash t \doteq \alpha_1 \times \alpha_2 \tag{1}$$

应用归纳假设可得 $i = 1, \ 2$

$$(\Gamma\theta \curlywedge (\alpha_i\theta /\!\!/ p_i)) \leq (\Gamma \curlywedge \Gamma_i)\theta \tag{2}$$

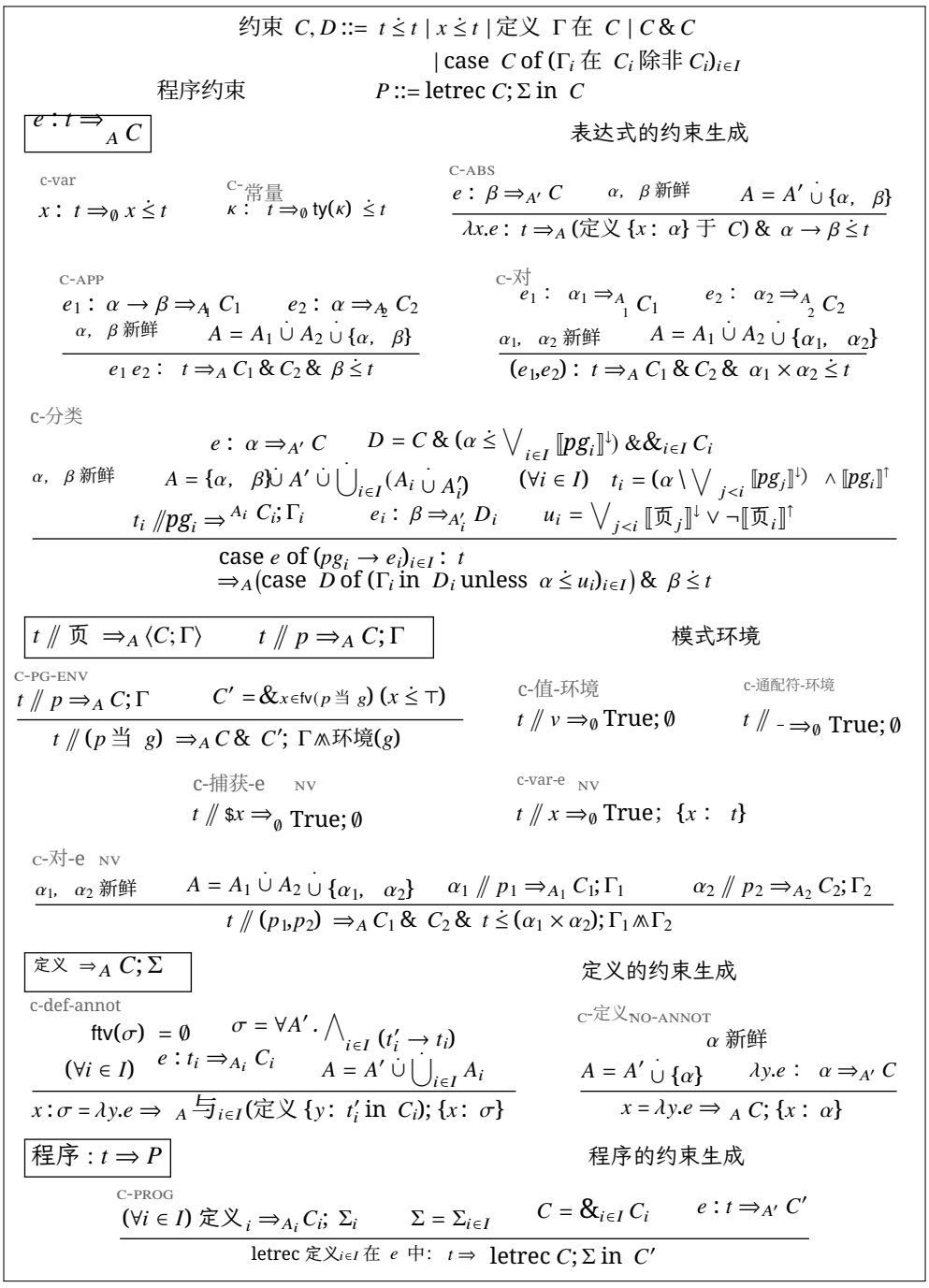

图14. 跟踪新鲜类型变量的约束生成

We have by definition $\Gamma^\star = t\theta \mathbin{/\!\!/} p = (\pi_1(t\theta) \mathbin{/\!\!/} p_1) \mathbin{\dot{\curlywedge}} (\pi_2(t\theta) \mathbin{/\!\!/} p_2)$. Then

$$\mathsf{dom}(\Gamma\theta \mathbin{\dot{\curlywedge}} \Gamma^\star) = \mathsf{dom}(\Gamma\theta \mathbin{\dot{\curlywedge}} \Gamma_1 \mathbin{\dot{\curlywedge}} \Gamma_2) = \mathsf{dom}((\Gamma \mathbin{\dot{\curlywedge}} \Gamma')\theta)$$

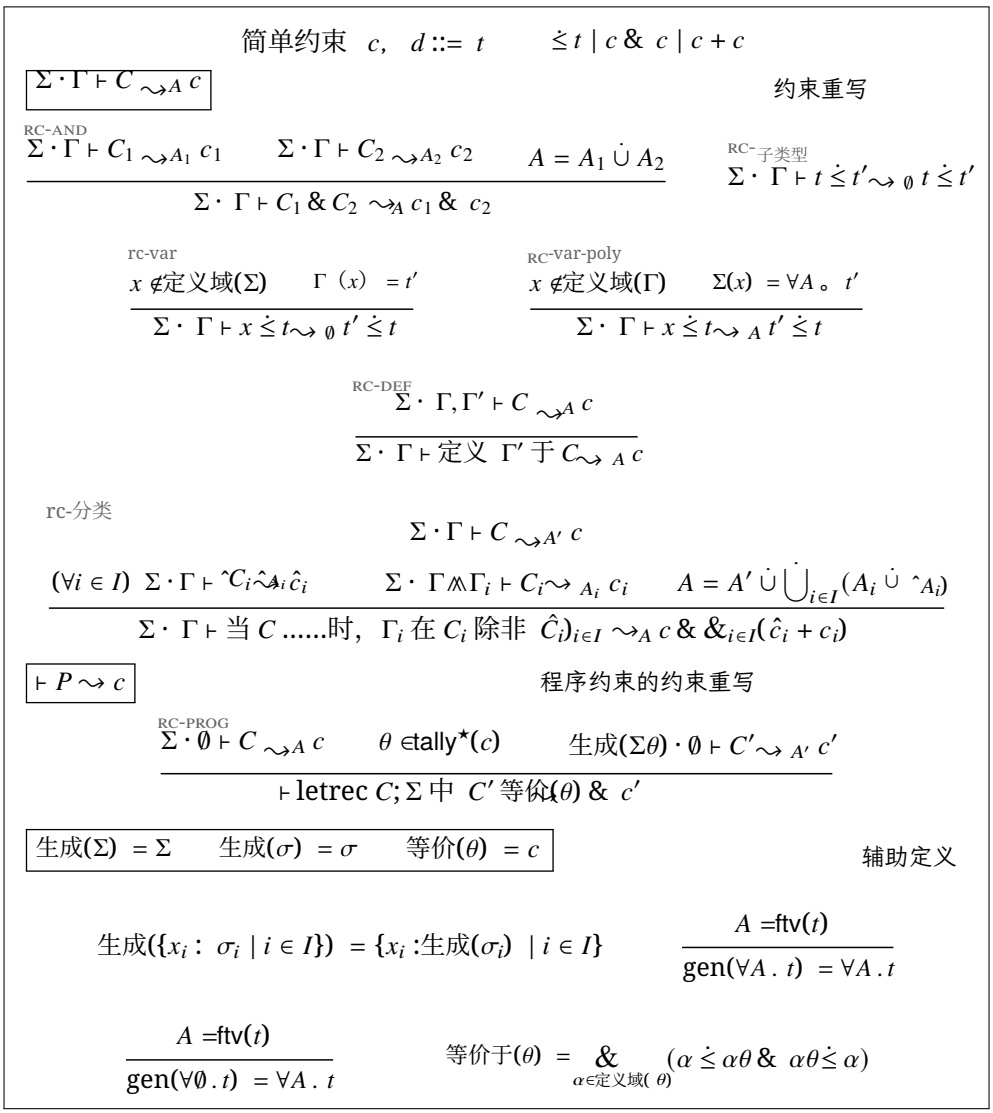

图 15. 约束重写与新鲜类型变量的跟踪

假设 $x \in \text{定义域}((\Gamma \Lsh \Gamma')\theta)$。则有

$$(\Gamma\theta \Lsh \Gamma^\star)(x) = (\Gamma\theta \Lsh (\pi_1(t\theta) /\!\!/ p_1) \Lsh (\pi_2(t\theta) /\!\!/ p_2))(x) \overset{\text{see below}}{\leq}$$

$$(\Gamma\theta \Lsh (\alpha_1\theta /\!\!/ p_1) \Lsh (\alpha_1\theta /\!\!/ p_2))(x) \overset{\text{by (2)}}{\leq} (\Gamma \Lsh \Gamma_1 \Lsh \Gamma_2)\theta(x) = (\Gamma \Lsh \Gamma')\theta(x)$$

如所要求的那样。标记为“见下文”的步骤由引理 A.28 以及以下事实推出：$\pi_i(t\theta) \leq \alpha_i\theta$。该事实成立，因为由（1）可得 $t\theta \leq \alpha_1\theta \times \alpha_2\theta$，因此根据性质 A.2（ii），有 $\pi_i(t\theta) \leq \alpha_i\theta$。

结束分类讨论。 □

引理 B.2（守卫模式的环境性质）。

(i) 定义域($t /\!\!/ p$) = pbv(p) (ii) 定义域($t /\!\!/ p$ when g) = pbv(p) ∪ fv(g) (iii) 若 $t /\!\!/ p \Rightarrow C; \Gamma$，则定义域($\Gamma$) = pbv($p$)。(iv) 若 $t /\!\!/ p$ when $g \Rightarrow C; \Gamma$，则定义域(Γ) = pbv(p) ∪ fv(g)。(v) ftv($[\![p]\!]^{\rho}_{\Gamma}$) ⊆ ftv($\Gamma$) 且 ftv(环境($g$)) = ftv($[\![\mathrm{pg}]\!]^{\uparrow}$) = ftv($[\![\mathrm{pg}]\!]^{\downarrow}$) = ∅

证明。

(i) 由引理 A.15(iii) 得出。(ii) 结合 (i) 以及 dom(env(g)) = fv(g) 这一事实得出。(iii) 通过对 p 的结构进行归纳得出。(iv) 结合 (iii) 以及 dom(env(g)) = fv(g) 这一事实得出。(v) ftv($[\![p]\!]^{\rho}_{\Gamma}$) ⊆ ftv($\Gamma$) 通过对 p 的结构进行归纳得出，这里用到了对所有 κ，ty(κ) 都是封闭的这一事实。ftv(env(g)) = ∅ 通过对 g 的结构进行归纳得出。ftv($[\![\mathrm{pg}]\!]^{\uparrow}$) = ftv($[\![\mathrm{pg}]\!]^{\downarrow}$) = ∅ 结合前述两个等式得出。□

引理 B.3. ⩓ 运算是结合的且交换的。

Proof. 由类型上的交运算符的结合律和交换律可得。□

引理 B.4（单个带守卫模式的环境生成的可靠性）。给定某个类型 t 和带守卫模式 pg。假设 $t /\!\!/ pg \Rightarrow \langle C;\ \Gamma' \rangle$ 且 $\Sigma \cdot \Gamma \vdash C \quad d$ 并且$\theta \Vdash d$ 对于某些 C、Σ、Γ、$\not\rightsquigarrow$ d和 θ成立。定义 $\Gamma^{\star} = t\theta /\!\!/ pg$。则 $\Gamma\theta \barwedge \Gamma^{\star} \leq (\Gamma \barwedge \Gamma')\theta$。

证明。假设 $pg = p$ 当 g时成立。我们有

$$\Gamma^{\star} = t\theta /\!\!/ pg = (t\theta /\!\!/ p) \barwedge \mathsf{env}(g)$$

此外，对用于推导 $t /\!\!/ pg \Rightarrow C; \Gamma'$ 的规则 C-PG-ENV 应用逆转规则：

$$\begin{aligned} t /\!\!/ p &\Rightarrow \langle C; \Gamma'' \rangle \\ \Gamma' &= \Gamma'' \barwedge \mathsf{env}(g) \end{aligned} \tag{1}$$

因此，由引理 B.1

$$(\Gamma\theta \barwedge (t\theta /\!\!/ p)) \leq (\Gamma \barwedge \Gamma'')\theta \tag{2}$$

因此，我们得到所需结果：

$$\Gamma\theta \barwedge \Gamma^{\star} = \Gamma\theta \barwedge (t\theta /\!\!/ p) \barwedge \mathsf{env}(g) \overset{(2) \text{ and } \textit{Lemma B.3}}{\leq} \Gamma\theta \barwedge \Gamma''\theta \barwedge \mathsf{env}(g) \overset{(1) \text{ and } \textit{Lemma B.2}(v)}{=} (\Gamma \barwedge \Gamma')\theta$$

□

引理 B.5（多个守卫模式的环境生成可靠性）。给定某个类型 t 和一系列带守卫的模式 $pg_{i \in I}$。假设 $t /\!\!/ pg_i \Rightarrow \langle C_i; \Gamma_i \rangle$ 以及$\Sigma \cdot \Gamma \vdash C_i \quad d_i$，并且对所有 $i \in I$以及某些 $C_{i \in I}$、$\Gamma_{i \in I}$、Σ、Γ、$d_{i \in I}$ 和 θ，都有$\theta \Vdash d_i$。

定义 $\Gamma' = \Gamma \barwedge \bigbarwedge_{i \in I} \Gamma_i$ 和 $\Gamma^{\star} = \Gamma\theta \barwedge \bigbarwedge_{i \in I} (t\theta /\!\!/ pg_i)$。那么 $\Gamma^{\star} \leq \Gamma'\theta$。

证明。对 $|I|$ 使用引理B.4进行归纳。□

定义 *B.6*。约束 C的自由表达式变量，记作$\mathsf{fv}(C)$，定义如下：

$$\mathsf{fv}(t \doteq t) = \emptyset$$
$$\mathsf{fv}(x \doteq t) = \{x\}$$
$$\mathsf{fv}(\mathtt{def}\ \Gamma\ \mathtt{in}\ C) = \mathsf{fv}(C) \setminus \mathsf{dom}(\Gamma)$$

自由变量(case C of $(\Gamma_i$ in C_i unless $\hat{}\ C_i)_{i\in I}$) = 自由变量(C) $\cup$ ($\cup_{i\in I}$自由变量($\hat{}\ C_i$)) $\cup$ ($\cup_{i\in I}$(自由变量(C_i) \ 定义域(Γ_i)))自由变量(C_1 & C_2) = 自由变量(C_1) $\cup$ 自由变量(C_2)

引理 B.7（约束生成和重写中的自由变量）。

(i) 如果 $\Sigma \cdot \Gamma \vdash C \rightsquigarrow d$，那么 $fv(C) \subseteq dom(\Sigma) \cup dom(\Gamma)$。 *(ii)* 如果 $e : t \Rightarrow C$，且 $\Sigma \cdot \Gamma \vdash C \rightsquigarrow d$，那么 $fv(e) \subseteq dom(\Sigma) \cup dom(\Gamma)$。

证明。 (i) 的证明采用对 C 的结构归纳法。(ii) 的证明采用对 e 的结构归纳法。两个证明都很直接。

引理 B.8（表达式的算法化类型推导的可靠性）。若 $e : t \Rightarrow C$ 且$\Sigma \cdot \Gamma \vdash C \rightsquigarrow d$ 且 $\theta \Vdash d$，则$\Sigma\theta \cdot \Gamma\theta \vdash e : t\theta$。

证明。 对 e 进行结构归纳。

分类讨论 的形式 e。

- 情形 $e = x$：于是 $C = x \doteq t$。我们分三种情况讨论：– $\Gamma(x) = t'$ 且 $x \notin \mathsf{dom}(\Sigma)$。在这种情况下，$\Sigma \cdot \Gamma \vdash C \rightsquigarrow d$ 是由规则 rc-var 推导得到的，因此 $d = t' \doteq t$。于是，$t'\theta \leq t\theta$，所以由规则 d-var 和 d-sub 可得 $\Sigma\theta \cdot \Gamma\theta \vdash e : t\theta$。– $\Sigma(x) = \forall A.t'$ 且 $x \notin \mathsf{dom}(\Gamma)$。在这种情况下，$\Sigma \cdot \Gamma \vdash C \rightsquigarrow d$ 是由规则 rc-var-poly 推导得到的，因此 $d = t' \doteq t$。于是，$t'\theta \leq t\theta$。通过对 A 中的变量重命名，我们可以假设 $(\Sigma\theta)(x) = \forall A.(t'\theta)$。由规则 d-var-poly 可得 $\Sigma\theta \cdot \Gamma\theta \vdash x : t'\theta$。现在由规则 d-sub 即可得证。– $x \in \mathsf{dom}(\Sigma)$ 且 $x \in \mathsf{dom}(\Gamma)$。不可能，因为没有规则可以推导出 $\Sigma \cdot \Gamma \vdash C \rightsquigarrow d$。
- 情形 $e = \kappa$：由规则D-CONST 可直接得出，因为$\mathsf{ty}(\kappa)$ 是封闭的。
- 情形 $e = \lambda x.e_1$：对新鲜的 α，β，通过逆转规则 C-ABS 可得：

$$\lambda x.e_1 : t \Rightarrow \underbrace{\mathtt{def}\ \{x : \alpha\}\ \mathtt{in}\ C_1 \ \&\ (\alpha \to \beta) \doteq t}_{=C}$$
$$e_1 : \beta \Rightarrow C_1 \tag{1}$$

逆转规则 RC-AND和RC-DEF可得

$$\Sigma \cdot \Gamma \vdash \mathtt{def}\ \{x : \alpha\}\ \mathtt{in}\ C_1 \rightsquigarrow d_1$$
$$d = d_1\ \&\ (\alpha \to \beta) \doteq t \tag{2}$$
$$\Sigma \cdot \Gamma, x : \alpha \vdash C_1 \rightsquigarrow d_1 \tag{3}$$

对 (1) 和 (3) 应用归纳假设可得 $\Sigma\theta \cdot \Gamma\theta, x : \alpha\theta \vdash e_1 : \beta\theta$，因此由规则 d-abs

$$\Sigma\theta \cdot \Gamma\theta \vdash e : \alpha\theta \to \beta\theta$$

结合 (2) 和假设 $\theta \Vdash d$，再结合 $\alpha\theta \to \beta\theta \leq t\theta$，由规则 D-SUB $\Sigma\theta \cdot \Gamma\theta \vdash e : t\theta$即可得证。

- 情况 $e = e_1\ e_2$：由归纳假设得出。

• 情况 $e = (e_1,\ e_2)$：由归纳假设得出。
• 情况 $e = \texttt{case}\ e_0$ 中 $(pg_i \to e_i)_{i\in I}$：我们有 情况 e_0 中 $(pg_i \to e_i)_{i\in I}$ ： $t \Rightarrow C$，因此根据逆转规则 c-分类

$$e_0 : \alpha \Rightarrow C_0 \tag{4}$$
$$C_0' = C_0\ \&\ \&_{i\in I}\, C_i''\ \&\ (\alpha \doteq \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow})$$
$$\alpha, \beta \text{ fresh}$$
$$(\forall i \in I) \quad t_i = (\alpha \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow}$$
$$t_i /\!\!/ pg_i \Rightarrow \langle C_i''; \Gamma_i\rangle \tag{5}$$
$$e_i : \beta \Rightarrow C_i' \tag{6}$$
$$u_i = \bigvee_{j<i} [\![pg_j]\!]^{\downarrow} \vee \neg [\![pg_i]\!]^{\uparrow}$$
$$C = (\texttt{case}\ C_0'\ \texttt{of}\ (\Gamma_i\ \texttt{in}\ C_i'\ \texttt{unless}\ \alpha \doteq u_i)_{i\in I})\ \&\ \beta \doteq t$$

进一步 $\Sigma \cdot\ \Gamma \vdash C \rightsquigarrow d$ 因此根据逆转规则 rc-case和 rc-and

$$\Sigma \cdot \Gamma \vdash C_0 \rightsquigarrow d_0 \tag{7}$$
$$(\forall i \in I)\ \Sigma \cdot \Gamma \vdash C_i'' \rightsquigarrow d_i'' \tag{8}$$
$$\Sigma \cdot \Gamma \between \Gamma_i \vdash C_i' \rightsquigarrow d_i' \tag{9}$$
$$d_0' = d_0\ \&\ \&_{i\in I}\, d_i''\ \&\ (\alpha \doteq \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow})$$
$$d_i = d_i' + \alpha \doteq u_i$$
$$d = d_0'\ \&\ \&_{i\in I}\, d_i\ \&\ \beta \doteq t$$

我们要证明的目标是

$$\Sigma\theta \cdot \Gamma\theta \vdash e_0 : \hat{t}_0 \text{ for some } \hat{t}_0 \tag{G1}$$
$$\hat{t}_0 \leq \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow} \tag{G2}$$
$$(\forall i \in I) \quad \hat{t}_i = (\hat{t}_0 \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow}$$
$$\mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma \between (\hat{t}_i /\!\!/ pg_i)) \tag{G3}$$
$$\mathsf{fv}(pg_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma) \tag{G4}$$
$$\begin{cases} \hat{t}_i' = \bot & \text{if } \hat{t}_i \leq \bot \\ \Sigma\theta \cdot \Gamma\theta \between (\hat{t}_i /\!\!/ pg_i) \vdash e_i : \hat{t}_i' & \text{otherwise} \end{cases} \tag{G5}$$
$$\bigvee_{i\in I} \hat{t}_i \leq t\theta \tag{G6}$$

该结论随后可由规则 D-CASE和 D-SUB推出。
(G1) 由 $\theta \Vdash d$ 以及 d的定义，我们有 $\theta \Vdash d_0$。因此，结合 (4)、(7) 和归纳假设，可得 $\Sigma\theta \cdot\ \Gamma\theta \vdash e_0 :\ \ \alpha\theta$。定义 $\hat{}t_0 = \alpha\theta$。这证明了 **(G1)**。
(G2) 根据 $\theta \Vdash d$ 和 d 的定义，d_0'可得 $\theta \Vdash \alpha \doteq \vee_{i\in I} [\![pg_i]\!]^{\downarrow}$。根据引理 B.2，我们知道 $\vee_{i\in I} [\![pg_i]\!]^{\downarrow}$ 是封闭的。因此，$\hat{}t_0 = \alpha\theta \leq \vee_{i\in I} [\![pg_i]\!]^{\downarrow}$，这证明了 **(G2)**。

(G3) 根据 (6) 和 (9)，结合引理 B.7 可得

$$\mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Gamma \twoheadrightarrow \Gamma_i) = \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Gamma_i)$$

假设 $pg_i = p_i$ 当 g_i，由（5）和引理 B.2 可得

$$\mathsf{dom}(\Gamma_i) = \mathsf{pbv}(p_i) \cup \mathsf{fv}(g_i) = \mathsf{dom}(\hat{t}_i /\!\!/ pg_i)$$

这证明了（G3）。

(G4) 由（5）逆转规则 c-pg-env

$$\mathsf{fv}(pg_i) \subseteq \mathsf{fv}(C_i'') \overset{\text{definition of } C_0', C}{\subseteq} \mathsf{fv}(C)$$

由假设 $\Sigma \cdot \Gamma \vdash C \quad d$ 和引理 B.7(i)，可得 $\mathsf{fv}(C)$ ⊆定义域(Σ) ∪定义域(Γ)。这证明了（G4）。

(G5) 现在假设 $i \in I$。由 $\theta \Vdash d$ 和 d的定义，可得 $\theta \Vdash d_i$ 对于$d_i = d_i' + \alpha \dot{\leq} u_i$。因此，$\theta \Vdash d_i'$或 $\theta \Vdash \alpha \dot{\leq} u_i$。现在我们证明这两种情况下（G5）都成立。

分类讨论关于 $\theta \Vdash \alpha \dot{\leq} u_i$ 是否成立。

– 情况 $\theta \Vdash \alpha \dot{\leq} u_i$：因此

$$\begin{aligned} \hat{t}_0 = \alpha\theta \leq u_i\theta &= (\vee_{j<i} [\![pg_j]\!]^{\downarrow} \vee \neg [\![pg_i]\!]^{\uparrow})\theta \\ &\overset{Lemma\ B.2}{=} (\vee_{j<i} [\![pg_j]\!]^{\downarrow} \vee \neg [\![pg_i]\!]^{\uparrow}) = u_i \end{aligned} \tag{10}$$

现在我们证明 $\hat{}t_i \leq \bot$，以便定义 $\hat{}t_i' = \bot$。我们有

$$\hat{t}_i = (\hat{t}_0 \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow}$$

我们从集合论角度进行论证。假设 $\xi \in \hat{}t_i$。那么 $\xi \in \hat{}t_0$and $\xi \notin \vee_{j<i} [\![$ pg $_j]\!]^{\downarrow}$and $\xi \in [\![\mathrm{pg}_i]\!]^{\uparrow}$。结合 (10)，得到 $\xi \in u_i$，因此根据 u_i 的定义，有 $\xi \in \vee_{j<i} [\![\mathrm{pg}\ _j]\!]^{\downarrow}$or $\xi \notin [\![\mathrm{pg}_i]\!]^{\uparrow}$。这就产生了矛盾，所以 $\xi \notin \hat{}t_i$，因此 $\hat{}t_i \leq \bot$。– 情况 not $\theta \Vdash \alpha \dot{\leq} u_i$：因此 $\theta \Vdash d_i'$。如果 $\hat{}t_i \leq \bot$，定义 $\hat{}t_i' = \bot$。否则，按如下方式继续。结合 (6)、(9) 和归纳假设，我们得到

$$\Sigma\theta \cdot (\Gamma \twoheadrightarrow \Gamma_i)\theta \vdash e_i : \beta\theta \tag{11}$$

定义

$$\Gamma_i^{\star} = \hat{t}_i /\!\!/ pg_i$$

根据 t_i 的定义，$\hat{}t_i$，$\hat{}t_0 = \alpha\theta$，并根据引理 B.2 可得 $t_i\theta = \hat{}t_i$。由 $\theta \Vdash d$ 以及 d 的定义，d_0'可得 $\theta \Vdash d_i''$。接着由（5）、（8）和引理 B.5可得

$$(\Gamma\theta \twoheadrightarrow \Gamma_i^{\star}) \leq (\Gamma \twoheadrightarrow \Gamma_i)\theta$$

由（11）和引理 A.8可得

$$\Sigma\theta \cdot \Gamma\theta \twoheadrightarrow \Gamma_i^{\star} \vdash e_i : \beta\theta$$

定义 $\hat{}t_i' = \beta\theta$。由 $\theta \Vdash d$ 以及 d 的定义可得 $\theta \Vdash \beta \dot{\leq} t$。因此 $\hat{t'}_i \leq t\theta$。

结束分类讨论。

在两种情况下，我们都定义了某个 $\hat{}t_i'$，它满足 (G5) 的条件。

(G6) 对于所有 $i \in I$，我们都有 $\hat{}t_i' \leq t\theta$。由此证明了 (G6)。

结束分类讨论。 □

引理 B.9（定义的环境生成的可靠性）。假设定义 $\not\in; \Sigma$ 和 $\Sigma' \cdot \emptyset \vdash C \quad d$ 对于某个 Σ' 满足 $\Sigma \supseteq \Sigma'$。如果 $\theta \Vdash d$，那么 $\Sigma'\theta \vdash$ 定义 良构。

证明。
分类讨论，形式为定义。

- 定义的分类 $= (x = \lambda y。 e)$：那么对于 α 新鲜的，通过逆转规则 c-def-no-annot

$$\lambda y.e : \alpha \Rightarrow C$$
$$\Sigma = \{x : \alpha\}$$

根据引理 B.8 则 $\Sigma'\theta \cdot \emptyset \vdash \lambda y。 e : \alpha\theta$。由 $\Sigma \subseteq \Sigma'$ 可得 $\Sigma'\theta(x) = \alpha\theta$。因此，$\Sigma'\theta \vdash$ 定义 成立 根据规则 D-DEF-NO-ANNOT。

- 定义的分类 $= (x : \sigma : \lambda y。 e)$：那么通过逆转规则c-def-annot

$$\begin{gathered}
\mathsf{ftv}(\sigma) = \emptyset \\
\sigma = \forall A \,.\, \bigwedge_{i \in I} (t_i' \to t_i) \\
(\forall i \in I) \quad e : t_i \Rightarrow C_i \\
C = \mathop{\&}_{i \in I} (\mathtt{def}\ \{y : t_i'\}\ \mathtt{in}\ C_i) \\
\Sigma = \{x : \sigma\}
\end{gathered} \tag{1}$$

此外，通过逆转规则 rc-and和 rc-def

$$\begin{gathered}
(\forall i \in I) \quad \Sigma' \cdot \{y : t_i'\} \vdash C_i \rightsquigarrow d_i \\
d = \mathop{\&}_{i \in I} d_i
\end{gathered} \tag{2}$$

定义 $\Sigma_i = \Sigma',\ y : t_i'$ 对所有 $i \in I$。那么，根据（1）、（2）以及引理 B.8

$$\Sigma_i\theta \cdot \emptyset \vdash e : t_i\theta$$

由 $\mathsf{ftv}(\sigma) = \emptyset$ 并假设 A 是新鲜的，可得 $t_i\theta = t_i$ 和 $t_i'\theta = t_i'$ 对所有 $i \in I$均成立。因此，$\Sigma'\theta,\ y : t_i' \cdot \emptyset \vdash e : t_i$，所以根据规则 定义 ok D-DEF-ANNOT，可得 $\Sigma'\theta \vdash$。分类讨论结束。

□

We n现在重述并证明第 3.5 节中关于算法化类型推导可靠性的定理。

定理 *3.6*（程序的算法化类型推导的可靠性）。给定程序 *prog*、某个类型 t、程序约束 P、简单约束 d以及某个类型替换 θ。假设 $prog : t \Rightarrow P$ 且 $\vdash P \quad d$ 且 $\theta \Vdash d$。那么 $\vdash prog : t\theta$。 $\rightsquigarrow$

证明。 假设 程序 $=$ letrec 定义$_{i \in I}$ in e。那么根据逆转规则 C-PROG，我们得到：

$$def_i \Rightarrow \langle C_i; \Sigma_i \rangle \quad (\forall i \in I) \tag{1}$$
$$\Sigma = \Sigma_{i \in I}$$
$$C = \&_{i \in I}\, C_i \tag{2}$$
$$e : t \Rightarrow C' \tag{3}$$
$$P = \mathtt{letrec}\ \langle C; \Sigma \rangle\ \mathtt{in}\ C'$$

根据逆转规则 RC-PROG：

$$\Sigma \cdot \emptyset \vdash C \rightsquigarrow c \tag{4}$$

$$\text{there exists } \theta^\star \in \mathsf{tally}^\star(c) \tag{5}$$

$$\mathsf{gen}(\Sigma\theta^\star) \cdot \emptyset \vdash C' \rightsquigarrow c' \tag{6}$$

$$d = \mathsf{equiv}(\theta^\star)\ \&\ c' \tag{7}$$

现在，通过反演规则 C-DEF-ANNOT和 C-DEF-NO-ANNOT，考察定义 定义$_{i\in I}$ 和环境 Σ，假设 $|I| = n$ 和 α_i 是新鲜的。注意，所有 σ_i 都是封闭的。定义环境 Σ' 和 Σ''，其中 $t_i = \alpha_i\theta^\star$ 对于 $i = 1, \ldots k$ 以及 $A_i = \mathsf{ftv}(t_i)$。

$$\begin{aligned}
def_{i\in I} &= (x_1 = e_1), && \ldots, (x_k = e_k), && (x_{k+1} : \sigma_{k+1} = e_{k+1}), \ldots, (x_n : \sigma_n = e_n)\\
\Sigma &= \{x_1 = \alpha_1, && \ldots, x_k = \alpha_k, && x_{k+1} = \alpha_{k+1}, && \ldots, x_n = \alpha_n\}\\
\Sigma' &= \{x_1 = t_1, && \ldots, x_k = t_k, && x_{k+1} = \sigma_{k+1}, && \ldots, x_n = \sigma_n\}\\
\Sigma'' &= \{x_1 = \forall A_1.t_1, && \ldots, x_k = \forall A_k.t_k, && x_{k+1} = \sigma_{k+1}, && \ldots, x_n = \sigma_n\}
\end{aligned}$$

然后 $\Sigma\theta^\star = \Sigma'$ 和$\mathsf{gen}(\Sigma') = \Sigma''$。

此外，环境(定义 i) $= \langle \Sigma'_i;\ \Sigma''_i \rangle$对于所有 $i \in I$满足 $\Sigma' = \Sigma'_{i\in I}$且 $\Sigma'' = \Sigma''_{i\in I}$。由 (5) 得到 $\theta^\star \Vdash c$。结合 (1)，在 (C 中的定义（2），以及 (4) 和引理 B.9，于是

$$\underbrace{\Sigma\theta^\star}_{=\Sigma'} \vdash def_i\ \mathsf{ok} \quad (\forall i \in I) \tag{8}$$

由（3）、（6）、（7）、假设 $\theta \Vdash d$以及引理 B.8，可得

$$\mathsf{gen}(\Sigma\theta^\star)\theta \cdot \emptyset \vdash e : t\theta$$

因为 $\mathsf{gen}(\Sigma\theta^\star)$ 是封闭的，并且 $\Sigma\theta^\star = \Sigma'$ 且 $\mathsf{gen}(\Sigma') = \Sigma''$，所以

$$\Sigma'' \cdot \emptyset \vdash e : t\theta$$

根据（8）和规则 D-PROG，则 $\vdash prog :\ t\theta$，如所需。 □

B.2 完备性

在本节中，我们证明，通过声明式系统可定型的程序，也可通过算法式系统定型。

引理 B.10（带守卫的模式产生的简单约束）。若 $t \mathbin{/\!\!/} pg \Rightarrow C; \Gamma$ 或$t \mathbin{/\!\!/} p \Rightarrow C; \Gamma$ 则 C 是一个简单约束 c。

证明。 由定义显然成立。 □

引理 B.11（模式环境生成的完备性）。 假设$t\theta \leq [\![p]\!]^{\rho}_{\Gamma^\star}$ 对 $\Gamma^\star$ 封闭。此外，$t \mathbin{/\!\!/} p \Rightarrow_A \langle C; \Gamma \rangle$ 对 A 新鲜。则 C 是一个简单约束，并且存在 θ'，使得 定义域(θ') $= A$ 且 $\theta_\circ \cup \theta' \Vdash C$，并且 $\Gamma(x)(\theta_\circ \cup \theta') \leq (t\theta \mathbin{/\!\!/} p)(x)$ 对所有 $x \in \mathsf{pbv}(p)$均成立。

证明。 对 p 作归纳。
分类讨论 的形式。

- 情形 $p = v$ 或 $p =_$或 $p = \$x$：则 $C = \mathsf{True}$，且 $\Gamma = \emptyset$ 并且 $A = \emptyset$。取 $\theta_\prime$ 为恒等代换，结论立即成立。
- 情形 $p = x$：那么 $C = \mathsf{True}$，且 $\Gamma = \{x :\ t\}$ 和 $A = \emptyset$。选择 $\theta_\prime$ 作为恒等代换，即可立即得出结论。

- 情形 $p = (p_1,\ p_2)$：应用逆转规则 c-对-环境 可得

$$\alpha_i \mathbin{/\!\!/} p_i \Rightarrow_{A_i} \langle C_i; \Gamma_i \rangle \quad i = 1, 2 \tag{1}$$
$$\alpha_1, \alpha_2 \text{ fresh}$$
$$A = A_1 \mathbin{\dot\cup} A_2 \mathbin{\dot\cup} \{\alpha_1, \alpha_2\}$$
$$C = C_1 \mathbin{\&} C_2 \mathbin{\&} t \mathbin{\dot\leq} \alpha_1 \times \alpha_2$$
$$\Gamma = \Gamma_1 \mathbin{\text{⩓}} \Gamma_2$$

定义 $\theta^\star := \theta_\circ \cup [\pi_i(t\theta)/\alpha_i \mid i = 1,\ 2]$。我们有假设 $t\theta \leq [\![p]\!]^\rho_{\Gamma^\star} = [\![p_1]\!]^\rho_{\Gamma^\star} \times [\![p_2]\!]^\rho_{\Gamma^\star}$。根据性质 A.2(ii)，于是 $\pi_i(t\theta) \leq [\![p_i]\!]^\rho_{\Gamma^\star}$ 对于 $i = 1,\ 2$。因此$\alpha_i\theta^\star \leq [\![p_i]\!]^\rho_{\Gamma^\star}$。所以我们可以将归纳假设应用于 (1)，并得到对于 $i = 1,\ 2$，有 C_i 是一个简单约束，且存在 θ_i 使得

$$\mathsf{dom}(\theta_i) = A_i$$
$$\theta^\star \mathbin{\dot\cup} \theta_i \Vdash C_i$$
$$\Gamma_i(x)(\theta^\star \mathbin{\dot\cup} \theta_i) \leq (\alpha_i\theta^\star \mathbin{/\!\!/} p_i)(x) \quad \text{for all } x \in \mathsf{pbv}(p_i) \tag{2}$$

现在定义

$$\theta' := \theta_1 \mathbin{\dot\cup} \theta_2 \mathbin{\dot\cup} [\pi_i(t\theta)/\alpha_i \mid i = 1, 2]$$

于是我们有

$$\mathsf{dom}(\theta') = A$$
$$\theta \mathbin{\dot\cup} \theta' \Vdash C_1 \mathbin{\&} C_2$$
$$\theta \mathbin{\dot\cup} \theta' \Vdash t \mathbin{\dot\leq} \alpha_1 \times \alpha_2$$

最后一个断言成立，因为

$$t(\theta \mathbin{\dot\cup} \theta') = t\theta \overset{\text{Property A.2(i)}}{\leq} \pi_1(t\theta) \times \pi_2(t\theta) = (\alpha_1 \times \alpha_2)(\theta \mathbin{\dot\cup} \theta')$$

取 $x \in \mathsf{pbv}(p)$。我们仍需证明 $(\Gamma_1 \mathbin{\text{⩓}} \Gamma_2)(x)(\theta_\circ \cup \theta') \leq (t\theta \mathbin{/\!\!/} p)(x)$。

- 假设 $x \in \mathsf{pbv}(p_1)$ 且 $x \notin \mathsf{pbv}(p_2)$。那么

$$\begin{aligned}(\Gamma_1 \mathbin{\text{⩓}} \Gamma_2)(x)(\theta \mathbin{\dot\cup} \theta') &\overset{\text{Lemma B.2}}{=} \Gamma_1(x)(\theta \mathbin{\dot\cup} \theta') = \Gamma_1(x)(\theta^\star \mathbin{\dot\cup} \theta_1) \\ &\overset{(2)}{\leq} (\alpha_1\theta^\star \mathbin{/\!\!/} p_1)(x) = (\pi_1(t\theta) \mathbin{/\!\!/} p_1)(x) \\ &\overset{\text{Lemma B.2}}{=} ((\pi_1(t\theta) \mathbin{/\!\!/} p_1) \mathbin{\text{⩓}} (\pi_2(t\theta) \mathbin{/\!\!/} p_2))(x) = (t\theta \mathbin{/\!\!/} p)(x)\end{aligned}$$

- 类似地，若 $x \notin \mathsf{pbv}(p_1)$ 且 $x \in \mathsf{pbv}(p_2)$。
- 假设 $x \in \mathsf{pbv}(p_1)$ 且 $x \in \mathsf{pbv}(p_2)$。那么

$$\begin{aligned}(\Gamma_1 \mathbin{\text{⩓}} \Gamma_2)(x)(\theta \mathbin{\dot\cup} \theta') &= \Gamma_1(x)(\theta^\star \mathbin{\dot\cup} \theta_1) \wedge \Gamma_2(x)(\theta^\star \mathbin{\dot\cup} \theta_2) \\ &\overset{(2)}{\leq} (\alpha_1\theta^\star \mathbin{/\!\!/} p_1)(x) \wedge (\alpha_2\theta^\star \mathbin{/\!\!/} p_2)(x) \\ &= (\pi_1(t\theta) \mathbin{/\!\!/} p_1)(x) \wedge (\pi_2(t\theta) \mathbin{/\!\!/} p_2)(x) \\ &= ((\pi_1(t\theta) \mathbin{/\!\!/} p_1) \mathbin{\text{⩓}} (\pi_2(t\theta) \mathbin{/\!\!/} p_2))(x) = (t\theta \mathbin{/\!\!/} p)(x)\end{aligned}$$

结束分类讨论。 □

引理 *B.12*（带守卫的模式环境生成的完备性）。假设 $t\theta \le [\![pg]\!]^{\uparrow}$，其中 $pg = p$ *when* g。此外，假设 $t /\!\!/ pg \Rightarrow_A \langle C;\Gamma\rangle$，其中 A 是新鲜的。则 C 是一个简单约束，并且存在 θ'，使得 $dom(\theta') = A$，且 $\theta \cup \theta' \Vdash C$ 和 $\Gamma(x)(\theta \dot\cup \theta') \le (t\theta /\!\!/ pg)(x)$ 对所有 $x \in pbv(p) \cup fv(g)$ 均成立。

证明。由 $t /\!\!/ pg \Rightarrow_A C;\Gamma$ 可知，存在某个 $\Gamma^\star$

$$t /\!\!/ p \Rightarrow_A \langle C;\Gamma^\star\rangle$$
$$\Gamma = \Gamma^\star \mathbin{⩓} \mathsf{env}(g)$$

如果 $[\![pg]\!]^{\uparrow} = \bot$ 那么 $t\theta = \bot$，否则 $[\![pg]\!]^{\uparrow} = [\![p]\!]^{\uparrow}{}_{\mathsf{env}(g)}$。在这两种情况下 $t\theta \le [\![p]\!]^{\uparrow}{}_{\mathsf{env}(g)}$。根据引理 B.2，可知 $\mathsf{env}(g)$ 是封闭的。然后根据引理 B.11，存在 θ' 使得

$$\mathsf{dom}(\theta') = A$$
$$\theta \dot\cup \theta' \Vdash C$$
$$\Gamma^\star(x)(\theta \dot\cup \theta') \le (t\theta /\!\!/ p)(x) \quad (\forall x \in \mathsf{pbv}(p)) \tag{1}$$

取 $x \in \mathsf{pbv}(p) \cup \mathsf{fv}(g)$。我们需要证明 $\Gamma(x)(\theta \dot\cup \theta') \le (t\theta /\!\!/ pg)(x)$。

- 假设 $x \notin \mathsf{dom}(\mathsf{env}(g)) = \mathsf{fv}(g)$。那么 $\Gamma(x) = \Gamma\bigstar(x)$ 且 $(t\theta /\!\!/ \mathrm{pg})(x) = (t\theta /\!\!/ p)(x)$。因此，结论可由 (1) 得出。.
- 假设 $x \in \mathsf{dom}(\mathsf{env}(g)) = \mathsf{fv}(g)$。定义 $\theta'' := \theta \cup \theta'$。– 如果 $x \notin \mathrm{pbv}(p)$，则 $(\Gamma \mathbin{⩓} \Gamma')(x)\theta'' = \mathrm{env}(g)(x)\theta''$ 引理 B.2(v)= $\mathrm{env}(g)(x) = (t\theta /\!\!/ \mathrm{pg})(x)$ – 如果 $x \in \mathrm{pbv}(p)$，则 $(\Gamma \mathbin{⩓} \Gamma')(x)\theta'' = (\Gamma\bigstar \mathbin{⩓} \Gamma')(x)\theta'' \wedge \mathrm{env}(g)(x)\theta''$ 引理 B.2(v)= $(\Gamma\bigstar \mathbin{⩓} \Gamma')(x)\theta'' \wedge \mathrm{env}(g)(x)$

and

$$(t\theta /\!\!/ pg)(x) = (t\theta /\!\!/ p)(x) \wedge \mathsf{env}(g)(x)$$

由此可得结论（1）。 □

引理 B.13（约束重写的可能性）。对于所有 Σ，Γ，以及 C 满足 $\mathsf{fv}(C) \subseteq \mathsf{dom}(\Sigma) \uplus \mathsf{dom}(\Gamma)$ 存在 c 且新鲜 A 满足 $\Sigma \cdot \Gamma \vdash C \rightsquigarrow_A c$。

证明。对 C的结构进行直接归纳。 □

引理 B.14（约束生成中的自由变量）。若 $e : t \Rightarrow C$ 则 $\mathsf{fv}(C) \subseteq \mathsf{fv}(e)$。

证明。对 C的结构进行直接归纳。 □

引理 B.15。设 I 为一个指标集，满足 $i \in I$；设 $u_{k\in I}$ 和 t 为类型。假设 $t \le \vee_{k\in I} u_k$，并定义 $t' = (\vee_{j<i} u_j) \vee \neg u_i$ 和 $t'' = (t \setminus \vee_{j<i} u_j) \wedge u_i$。若 $t'' \le \bot$，则 $t \le t'$。

证明。我们从集合论角度进行论证。假设 $\xi \in t$ 但 $\xi \notin t'$。那么 $\xi \notin \vee_{j<i} u_j$ 且 $\xi \notin \neg u_i$。因此，$\xi \in u_i$。然而，$\xi \in t''$。这与 $t'' \le \bot$ 矛盾。□

引理 B.16（表达式算法化类型推导的完备性）。 给定 Σ、 Γ、 e、 t和 θ。如果 $\Sigma\theta \cdot \Gamma\theta \vdash e : t\theta$，那么存在 C、 d、新鲜类型变量集 A_1、 A_2 和 θ'，使得 $..e : t \Rightarrow_{A_1} C$，且 $\Sigma \cdot \Gamma \vdash C \quad_{A_2} d$ 并且 $\theta \cup \theta' \Vdash d$ 和 定义域$(\theta') = A_1 \cup A_2$。

证明。 通过对 e 进行结构归纳
分类讨论关于 e 的形式。

- 情形 $e = x$：根据规则 c-var

$$x : t \Rightarrow_{\emptyset} \underbrace{x \mathrel{\dot{\leq}} t}_{=C}$$

根据引理 A.6 可得 $\Sigma\theta \cdot \Gamma\theta \vdash e : t'$，其推导以规则 D-VAR 或D-VAR-POLY之一结束。此外， $t' \leq t\theta$。

– 若最后一条规则是 d-var，则 $\Gamma(x) = t''$，其中 $t''\theta = t'$ 且 $x \notin$ 定义域(Σ)。那么根据规则 rc-var

$$\Sigma \cdot \Gamma \vdash x \mathrel{\dot{\leq}} t \rightsquigarrow_{\emptyset} \underbrace{t'' \mathrel{\dot{\leq}} t}_{=:d}$$

令 θ' 为恒等代换。则 定义域$(\theta') = \emptyset$，显然。可以验证 $\theta \cup \theta' \Vdash d$。

– 若最后一条规则是 d-var-poly，则 $\Sigma(x) = \forall A_2 。 u$ 对于某个 u 和 A_2，其中 。 $t' = u(\theta \cup \theta')$ 对于某个 θ'，其中 定义域$(\theta') = A_2$。此外， $x \notin$ 定义域(Γ)。因此，应用规则 rc-var-poly

$$\Sigma \cdot \Gamma \vdash x \mathrel{\dot{\leq}} t \rightsquigarrow_{A_2} \underbrace{u \mathrel{\dot{\leq}} t}_{=:d}$$

..并且我们有 $u(\theta \cup \theta') = t' \leq t\theta = t(\theta \cup \theta')$，所以 $\theta \cup \theta' \Vdash d$。

- 情形 $e = \kappa$：由引理A.6 $\Sigma\theta \cdot \Gamma\theta \vdash \kappa :$ $\mathsf{ty}(\kappa)$ 和 $\mathsf{ty}(\kappa) \leq t\theta$。应用规则 c-const和 rc-subty

$$\kappa : t \Rightarrow_{\emptyset} \mathsf{ty}(\kappa) \mathrel{\dot{\leq}} t$$
$$\Sigma \cdot \Gamma \vdash \mathsf{ty}(\kappa) \mathrel{\dot{\leq}} t \rightsquigarrow_{\emptyset} \underbrace{\mathsf{ty}(\kappa) \mathrel{\dot{\leq}} t}_{=:d}$$

取 θ' 为恒等代换，并且 $A_1 = A_2 = \emptyset$。注意到$\mathsf{ty}(\kappa)$ 是封闭的，。我们有 $\theta \cup \theta' \Vdash d$。

- 情形 $e = \lambda x 。 e$：由引理A.6，通过逆转规则D-ABS，我们得到：

$$\Sigma\theta \cdot \Gamma\theta, x : t_1 \vdash e' : t_2 \tag{1}$$
$$t_1 \to t_2 \leq t\theta \tag{2}$$

选择新鲜的 α， β 并定义

$$\theta_1 := \theta \dot{\cup} [t_1/\alpha, t_2/\beta] \tag{3}$$

然后根据（1）： $\Sigma\theta_1 \cdot (\Gamma,\ x :\ \alpha)\ \theta_1 \vdash e' :\ \beta\theta_1$。现在应用归纳假设可得存在 θ_2 并满足

$$e' : \beta \Rightarrow_{A_1'} C' \quad (A_1' \text{ fresh}) \tag{4}$$

$$\Sigma \cdot \Gamma, x : \alpha \vdash C' \rightsquigarrow_{A_2} d' \quad (A_2 \text{ fresh}) \tag{5}$$

$$\theta_1 \dot\cup \theta_2 \Vdash d' \tag{6}$$

$$\mathsf{dom}(\theta_2) = A_1' \dot\cup A_2 \tag{7}$$

由（4）和规则 C-ABS，则

$$\lambda x.e' : t \Rightarrow_{A_1} \underbrace{\mathtt{def}\ \{x : \alpha\}\ \mathtt{in}\ C' \ \&\ (\alpha \to \beta) \dot\leq t}_{=:C} \tag{8}$$

$$A_1 = A_1' \dot\cup \{\alpha, \beta\} \tag{9}$$

由（5）和规则 RC-DEF 现在 $\Sigma \cdot \Gamma \vdash$ 定义 $\{x :\ \alpha\}$ 在 $C' \rightsquigarrow_{A_2} d'$。因此，由（8）则

$$\Sigma \cdot \Gamma \vdash C \rightsquigarrow_{A_2} \underbrace{d' \ \&\ (\alpha \to \beta) \dot\leq t}_{=:d} \tag{10}$$

由（8）和（10），我们只需证明存在 θ' 使得

$$\theta \dot\cup \theta' \Vdash d \tag{G1}$$

$$\mathsf{dom}(\theta') = A_1 \dot\cup A_2 \tag{G2}$$

定义

$$\theta' = [t_1/\alpha, t_2/\beta] \dot\cup \theta_2 \tag{11}$$

于是我们得到目标（G2）

$$\mathsf{dom}(\theta') = \{\alpha, \beta\} \dot\cup \mathsf{dom}(\theta_2) \overset{(7)}{=} A_1' \dot\cup \{\alpha, \beta\} \dot\cup A_2 \overset{(9)}{=} A_1 \dot\cup A_2 \tag{12}$$

此外，我们有

$$\theta \dot\cup \theta' = \theta \dot\cup [t_1/\alpha, t_2/\beta] \dot\cup \theta_2 \overset{(3)}{=} \theta_1 \dot\cup \theta_2$$

。由 (6) 得到 $\theta_1 \cup \theta_2 \Vdash d'$，因此根据 (10) 中 d 的定义，我们只需验证 $(\alpha \to \beta)(\theta \cup \theta') \leq t(\theta \cup \theta')$，即可证明 (G1)。根据 (11) 中 θ' 的定义，我们知道 $(\alpha \to \beta)(\theta \cup \theta') = t_1 \to t_2$。由 (7)、(12) 以及 α, β, A_1', A_2 的新鲜性，我们知道 $\mathsf{ftv}(t) \cap \mathsf{dom}(\theta') = \emptyset$。因此，$t(\theta \cup \theta') = t\theta$，并且 $(t_1 \to t_2) \leq t\theta$，由 (2) 可得。

- 情形 $e = e_1\ e_2$：根据归纳假设以及规则 C-APP 和 RC-AND 可得。
- 情形 $e = (e_1,\ e_2)$：由归纳假设以及规则 C-对 和 RC-AND可得。

- 情形 $e = \texttt{case}\ \ e_0\ \texttt{of}\ (pg_i \to e_i)_{i\in I}$：由引理 A.6，通过逆转规则D-CASE，我们得到：

$$\Sigma\theta \cdot \Gamma\theta \vdash e_0 : t_0\theta \tag{13}$$

$$t_0 \leq \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow} \tag{14}$$

$$(\forall i \in I) \quad t_i = (t_0 \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow} \tag{15}$$

$$\begin{cases} t_i' = \bot \text{ and } \mathsf{fv}(e_i) \subseteq \mathsf{dom}(\Sigma\theta) \cup \mathsf{dom}(\Gamma\theta \between (t_i /\!\!/ pg_i)) & \text{if } t_i \leq \bot \\ \Sigma\theta \cdot \Gamma\theta \between (t_i /\!\!/ pg_i) \vdash e_i : t_i' & \text{otherwise} \end{cases} \tag{16}$$

$$\mathsf{fv}(pg_i) \subseteq \mathsf{dom}(\Sigma) \cup \mathsf{dom}(\Gamma)$$

$$\Sigma\theta \cdot \Gamma\theta \vdash e : \bigvee_{i\in I} t_i'$$

$$t' := \bigvee_{i\in I} t_i' \leq t\theta \tag{17}$$

我们根据规则 c-分类 推导：

$$e : t \Rightarrow_{A_1} \underbrace{(\texttt{case}\ C_0'\ \texttt{of}\ (\Gamma_i''\ \texttt{in}\ D_i\ \texttt{unless}\ \alpha \dot{\leq} u_i')_{i\in I})\ \&\ \beta \dot{\leq} t}_{=C} \tag{18}$$

$$A_1 = \{\alpha, \beta\} \dot\cup B_0 \dot\cup \dot\bigcup_{i\in I} (B_i \dot\cup B_i') \tag{19}$$

$$e_0 : \alpha \Rightarrow_{B_0} C_0 \qquad (\alpha \text{ fresh}) \tag{20}$$

并且对所有 $i \in I$

$$u_i = (\alpha \setminus \bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \wedge [\![pg_i]\!]^{\uparrow}$$

$$u_i /\!\!/ pg_i \Rightarrow_{B_i} \langle C_i; \Gamma_i \rangle \tag{21}$$

$$e_i : \beta \Rightarrow_{B_i'} D_i \tag{22}$$

$$u_i' = (\bigvee_{j<i} [\![pg_j]\!]^{\downarrow}) \vee \neg [\![pg_i]\!]^{\uparrow} \tag{23}$$

所有类型变量集 B_0，B_i，B_i' 均假定为新鲜。式 C_0'（18）定义如下：

$$C_0' = C_0\ \&\ \&_{i\in I}\, C_i\ \&\ (\alpha \dot{\leq} \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow}) \tag{24}$$

定义

$$\theta^\star := \theta \dot\cup [t_0/\alpha] \tag{25}$$

然后，我们可以将归纳假设应用于（13），并结合（20）得到

$$\Sigma \cdot \Gamma \vdash C_0 \rightsquigarrow_{B_0'} d_0$$

$$\theta^\star \dot\cup \theta_0' \Vdash d_0 \tag{26}$$

$$\text{for some } \theta_0' \text{ with } \mathsf{dom}(\theta_0') = B_0 \dot\cup B_0' \tag{27}$$

根据引理 B.10 和（21），我们知道 C_i 对于所有 $i \in I$都是简单约束。因此，结合（24），并且由于简单约束可重写为简单约束，我们得到

$$\Sigma \cdot \Gamma \vdash C_0' \rightsquigarrow_{B_0'} \underbrace{d_0 \,\&\, \&_{i\in I}\, C_i \,\&\, (\alpha \dot{\leq} \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow})}_{=d_0'} \tag{28}$$

对于每个分支 $i \in I$（注意，$\mathsf{fv}(\vee_{j<i}[\![pg_j]\!]^{\downarrow}) = \emptyset$和$\mathsf{fv}([\![pg_i]\!]^{\uparrow}) = \emptyset$根据引理B.2（v）），根据 t_i 在（15）中的定义可得：

$$u_i\theta^\star = t_i \leq [\![pg_i]\!]^{\uparrow}$$

假设$pg_i = p_i$ 当 g_i时。根据引理B.12 和（21），存在$\theta_i^\star$使得

$$\mathsf{dom}(\theta_i^\star) = B_i \tag{29}$$

$$\theta^\star \dot{\cup} \theta_i^\star \Vdash C_i \tag{30}$$

$$\forall x \in \mathsf{pbv}(p_i) \cup \mathsf{fv}(g_i) : \\ (\Gamma_i \between \Gamma_i')(x)(\theta^\star \dot{\cup} \theta_i^\star) \leq \underbrace{(t_i /\!\!/ pg_i)(x)}_{=:\Gamma_i^\star} \tag{31}$$

根据 $\theta^\star$ 在（25）中的定义以及（14），可得

$$\theta^\star \Vdash \alpha \dot{\leq} \bigvee_{i\in I} [\![pg_i]\!]^{\downarrow}$$

因此，根据 d_0' 在（28）中的定义以及（26），可得（30）

$$\underbrace{\theta^\star \dot{\cup} \theta_0' \dot{\cup} \dot{\bigcup_{i\in I}} \theta_i^\star}_{=:\theta^{\star\star}} \Vdash d_0' \tag{32}$$

定义

$$\theta^\top := \theta^{\star\star} \dot{\cup} [t'/\beta] \tag{33}$$

假设 $i \in I$。现在我们证明以下三个目标（G1）、（G2）、（G3）：

$$\Sigma \cdot \Gamma \between \Gamma_i \vdash D_i \rightsquigarrow_{B_i''} d_i \quad \text{for some } d_i \text{ and fresh } B_i'' \tag{G1}$$

$$\text{there exists } \theta_i' \text{ with } \mathsf{dom}(\theta_i') = B_i' \dot{\cup} B_i'' \tag{G2}$$

$$\theta' \dot{\cup} \theta_i' \Vdash d_i \text{ or } t_i \leq \bot \tag{G3}$$

针对是否满足进行分类讨论。

– 不满足的情形不 $t_i \leq \bot$：根据（16）以及（$\Gamma_i^\star$）中的定义（31）：

$$\Sigma\theta \cdot (\Gamma\theta \between \Gamma_i^\star) \vdash e_i : t_i' \tag{34}$$

假设以下目标（G4），我们很快将对其进行证明：

$$(\Gamma \between \Gamma_i)\theta^\top \leq \Gamma\theta \between \Gamma_i^\star \tag{G4}$$

由引理 A.8 及规则 d-sub

$$\Sigma\theta \cdot (\Gamma \between \Gamma_i)\theta^\top \vdash e_i : \beta\theta^\top \tag{35}$$

结合 (22)、(35 和归纳假设，可知存在某个 θ_i'

$$\Sigma \cdot \Gamma \between \Gamma_i \vdash D_i \rightsquigarrow_{B_i''} d_i$$
$$\theta^\top \dot\cup \theta_i' \Vdash d_i$$
$$\mathsf{dom}(\theta_i') = B_i' \dot\cup B_i''$$

为了完成情形 $t_i \not\leq \bot$，我们还需要证明 (G4)。也就是说，我们需要证明 $(\Gamma \between \Gamma_i)\theta^\top \leq \Gamma\theta \between \Gamma_i^\star$。根据引理B.2 以及 $\theta_i^\star$和 θ_i'的定义，我们有定义域$(\Gamma_i^\star) = \mathsf{pbv}(p_i) \cup \mathsf{fv}(g_i) =$定义域$(\Gamma_i)$。现在假设 $x \in$定义域$(\Gamma) \cup$定义域$(\Gamma_i^\star)$。我们需要证明 $(\Gamma \between \Gamma_i)\theta^\top(x) \leq (\Gamma\theta \between \Gamma_i^\star)(x)$。

* 如果 $x \in$定义域(Γ) 且 $x \notin$定义域$(\Gamma_{\star i}) =$定义域(Γ_i)，那么

$$(\Gamma \between \Gamma_i)\theta^\top(x) = \Gamma(x)\theta^\top = \Gamma(x)\theta = (\Gamma\theta \between \Gamma_i^\star)(x)$$

第二个等式成立，因为

$$\begin{aligned}\theta^\top = \theta^{\star\star} \dot\cup [t'/\beta] = \theta^\star \dot\cup \theta_0' \cup (\dot\bigcup_{j\in I} \theta_j^\star) \dot\cup [t'/\beta] = \\ \theta \dot\cup \theta_0' \cup (\dot\bigcup_{j\in I} \theta_j^\star) \dot\cup [t_0/\alpha, t'/\beta]\end{aligned} \tag{36}$$

并且定义域(θ_0')，定义域$(\circ_{\cup_{j\in I}} \theta_j^\star)$ 中的所有类型变量，以及 α，β 都是新鲜的。

* 若 $x \in$定义域(Γ) 且 $x \in$定义域$(\Gamma_{\star i}) =$定义域(Γ_i)，则

$$\begin{aligned}(\Gamma \between \Gamma_i)\theta^\top(x) &= \text{by (36), freshness of type variables, definition of } \between \\ \Gamma(x)\theta \wedge \Gamma_i(x)\theta^\top &\leq \text{by (31)} \\ \Gamma(x)\theta \wedge \Gamma_i^\star(x) &= \\ (\Gamma\theta \between \Gamma_i^\star)(x)\end{aligned}$$

* 若 $x \notin$定义域(Γ) 且 $x \in$定义域$(\Gamma_{\star i}) =$定义域(Γ_i)，则

$$(\Gamma \between \Gamma_i)\theta^\top(x) = \Gamma_i(x)\theta^\top \overset{(31)}{\leq} \Gamma_i^\star(x) = (\Gamma\theta \between \Gamma_i^\star)(x)$$

这就完成了 (G4) 的证明，因此该情形 $t_i \not\leq \bot$也得证。

– 情形 $t_i \leq \bot$：我们有

$$\mathsf{fv}(D_i) \overset{(22),\textit{Lemma B.14}}{\subseteq} \mathsf{fv}(e_i) \overset{(16)}{\subseteq} \mathsf{dom}(\Sigma\theta) \cup \mathsf{dom}(\Gamma\theta \between (t_i /\!\!/ pg_i)) \overset{(21),\textit{Lemma B.2}}{=} \mathsf{dom}(\Sigma\theta) \cup \mathsf{dom}(\Gamma \between \Gamma_i)$$

根据引理 B.13 则

$$\Sigma \cdot \Gamma \between \Gamma_i \vdash D_i \rightsquigarrow_{B_i''} d_i$$

对于某个 d_i，且 B_i'' 是新鲜的。在这种情况下，定义

$$\theta_i' := [\bot/\alpha \mid \forall \alpha \in B_i' \dot\cup B_i'']$$

分类讨论结束。这就完成了目标 (G1)、(G2) 和 (G3) 的证明。

由（18），我们得到 $e:\ t \Rightarrow_{A_1} C$。根据（18）中 C 的定义，结合（28）、(G1) 以及规则 RC-CASE，我们得到

$$\Sigma \cdot \Gamma \vdash C \rightsquigarrow_{A_2} \underbrace{d'_0 \mathbin{\&} \mathop{\&}_{i\in I}(d_i + \alpha \dot{\leq} u'_i)}_{=:d} \tag{37}$$

$$A_2 = B'_0 \dot{\cup} \dot{\bigcup_{i\in I}} B''_i \tag{38}$$

我们将 θ' 定义如下：

$$\theta' =: [t_0/\alpha, t'/\beta] \dot{\cup} \theta'_0 \dot{\cup} \dot{\bigcup_{i\in I}} (\theta'_i \dot{\cup} \theta^\star_i) \tag{39}$$

并验证 $\theta \dot{\cup} \theta' \Vdash d$ 和 $\text{定义域}(\theta') = A_1 \cup A_2$。我们有

$$\theta \dot{\cup} \theta' \overset{(25)}{=} \theta^\star \dot{\cup} [t'/\beta] \dot{\cup} \theta'_0 \dot{\cup} \dot{\bigcup_{i\in I}} (\theta'_i \dot{\cup} \theta^\star_i) \overset{(32)}{=} \theta^{\star\star} \dot{\cup} [t'/\beta] \dot{\cup} \dot{\bigcup_{i\in I}} \theta'_i$$
$$\overset{(33)}{=} \theta^\top \dot{\cup} \dot{\bigcup_{i\in I}} \theta'_i \tag{40}$$

于是，对所有 $i \in I$，我们有

$$\theta \dot{\cup} \theta' \Vdash d_i + (\alpha \dot{\leq} u'_i) \tag{41}$$

因为.....

- ... 如果 $t_i \not\leq \bot$ 那么 $\theta \dot{\cup} \theta' \Vdash d_i$ 根据 (G3) 和 (40)。
- ..... 如果 $t_i \leq \bot$ 那么 $\theta \dot{\cup} \theta' \Vdash \alpha \dot{\leq} u'_i$ 根据以下推理：

$$\alpha(\theta \dot{\cup} \theta') = t_0 \overset{(14)}{\leq} \bigvee_{i\in I} [\![pg_i]\!]^\downarrow$$
$$u'_i(\theta \dot{\cup} \theta') \overset{(23) \text{ and } \textit{Lemma B.2}}{=} (\bigvee_{j<i} [\![pg_j]\!]^\downarrow) \vee \neg [\![pg_i]\!]^\uparrow$$
$$t_i \overset{(15)}{=} (t_0 \setminus \bigvee_{j<i} [\![pg_j]\!]^\downarrow) \wedge [\![pg_i]\!]^\uparrow$$

于是，结合 $t_i \leq \bot$ 由引理 B.15

$$\alpha(\theta \dot{\cup} \theta') \leq u'_i = u'_i(\theta \dot{\cup} \theta')$$

。此外，由 (32) 和 (40) 可得 $\theta \cup \theta' \Vdash d'_0$。因此，根据 (d 中的定义37) 以及 (41)，可得

$$\theta \dot{\cup} \theta' \Vdash d$$

. 如所需。接下来需要证明 $\text{dom}(\theta') = A_1 \cup A_2$。由 (G2) 可得 $\text{dom}(\theta'_i) = B''_i \cup B'''_i$ 并且由 (29) 可得 $\text{dom}(\theta^\star_i) = B_i \cup B'_i$ 对所有 $i \in I$。由 (27) 可得 $\text{dom}(\theta'_0) = B_0 \cup B'_0$。根据 (39) 中 θ' 的定义，可得

$$\text{dom}(\theta') = \{\alpha, \beta\} \dot{\cup} B_0 \dot{\cup} B'_0 \dot{\cup} \dot{\bigcup_{i\in I}} (B_i \dot{\cup} B'_i \dot{\cup} B''_i)$$

。最后，由 (19) 和 (38) 可得 $A_1 \cup A_2 = \text{定义域}(\theta')$。

结束分类讨论。 □

定义 *B.17.* 关系 定义 $\Rightarrow \Sigma$ 和 定义$_{i\in I} \Rightarrow \Sigma$ 由以下推理规则定义：

$$\frac{\alpha \text{ fresh}}{(x = e) \Rightarrow \{x : \alpha\}} \qquad \frac{\mathsf{ftv}(\sigma) = \emptyset}{(x : \sigma = e) \Rightarrow \{x : \sigma\}} \qquad \frac{(\forall i \in I)\ \ def_i \Rightarrow \Sigma_i}{def_{i\in I} \Rightarrow \Sigma_{i\in I}}$$

引理 B.18（单个定义的环境生成完备性）。假设对于某个 Σ，有 $def \Rightarrow \Sigma'$ 和 $\Sigma' \subseteq \Sigma$。若 $\Sigma\theta \vdash def\,\mathsf{ok}$，则 $def \Rightarrow_{A_1} \langle C; \Sigma'\rangle$ 且 $. .\Sigma \cdot \emptyset \vdash C \quad {}_{A_2} d$，并且存在 θ'，满足 $\theta \cup \theta' \Vdash\!\!\rightarrow d$ 且 定义域$(\theta') = (A_1 \cup A_2) \setminus \mathsf{ftv}(\Sigma')$。

证明。
根据 *def* 的形式进行分类讨论。

- 情形 $def = (x = \lambda y。e)$：已知 $def \Rightarrow \Sigma'$，有 $\Sigma' = \{x : \alpha\}$，其中 α。根据 $\Sigma' \subseteq \Sigma$，得到 $\Sigma(x) = \alpha$。根据假设 $\Sigma\theta \vdash def\,\mathsf{ok}$，通过逆转规则d-def-no-annot

$$\Sigma\theta \cdot \emptyset \vdash \lambda y.e : \alpha\theta$$

根据引理B.16，存在 θ'，使得

$$\lambda y.e : \alpha \Rightarrow_{A_1'} C$$
$$\Sigma \cdot \emptyset \vdash C \rightsquigarrow_{A_2} d$$
$$\theta \mathbin{\dot\cup} \theta' \Vdash d$$
$$\mathsf{dom}(\theta') = A_1' \mathbin{\dot\cup} A_2$$

。使用规则 C-DEF-NO-ANNOT 同样可以定义 $\Rightarrow_{A_1} C; \Sigma'$，其中 $A_1 = \{\alpha\} \cup A_1'$。

- 情况定义 $= (x : \sigma = \lambda y。e)$：由 $\Sigma\theta \vdash$ 定义 成立 通过逆转规则d-def-annot

$$\Sigma\theta(x) = \sigma = \Sigma(x)$$
$$\mathsf{ftv}(\sigma) = \emptyset \tag{1}$$
$$\sigma = \forall A . \bigwedge_{i\in I} (t_i' \to t_i)$$
$$(\forall i \in I) \quad \Sigma\theta \cdot \{y : t_i'\} \vdash e : t_i$$

因此，取 A 新鲜 $\Sigma\theta \cdot \{y : t_i'\}\theta \vdash e : t_i\theta\,(\forall i \in I)$。因此，根据引理 B.16 对所有 $i \in I$：

$$e : t_i \Rightarrow_{B_i} C_i$$
$$\Sigma \cdot y : t_i' \vdash C_i \rightsquigarrow_{B_i'} d_i \tag{2}$$
$$\theta \mathbin{\dot\cup} \theta_i \Vdash d_i \tag{3}$$
$$\mathsf{dom}(\theta_i) = B_i \mathbin{\dot\cup} B_i' \tag{4}$$

此外，$\Sigma' = \{x : \sigma\}$ 根据假设 定义 $\Rightarrow \Sigma'$。因此，使用规则 c-def-annot

$$def \Rightarrow_{A_1} \langle C; \Sigma'\rangle$$
$$C = \&_{i\in I}(\mathtt{def}\ \{y : t_i'\}\ \mathtt{in}\ C_i)$$
$$A_1 = \dot{\bigcup_{i\in I}} B_i$$

并且结合（2）和规则 rc-def以及 rc-and

$$\Sigma \cdot \emptyset \vdash C \rightsquigarrow_{A_2} d$$
$$d = \&_{i \in I} d_i$$
$$A_2 = \dot{\bigcup}_{i \in I} B'_i$$

.. .定义 $\theta' = \bigcup_{i \in I} \theta_i$。然后 $\theta \cup \theta' \Vdash d$ 根据 (3)。并结合 (4)，定义域$(\theta') = A_1 \cup A_2$。注意，$\mathrm{ftv}(\Sigma') = \emptyset$ 来自 (1)。

结束分类讨论。 □

引理 B.19（多个定义的环境生成完备性）。假设 $\Sigma = \Sigma_{i \in I}$ 且 $def_{i \in I} \Rightarrow \Sigma$ 且 $\Sigma\theta \vdash def_i$ 对所有 都成立 。那么 $def_i \Rightarrow_{A_i} \langle C_i;\ \Sigma_i \rangle$ 对所有 $i \in I$ 以及某个 C_i 都成立。此外，$\Sigma \cdot \emptyset \vdash \&_{i \in I} C_i \quad_A d$，并且存在 θ'，使得 $\theta_\circ \cup \theta' \Vdash d$ 且 .. 定义域 $(\theta') = (A \cup \bigcup_{i \in I} A_i) \setminus \mathrm{ftv}(\Sigma)$。

证明。通过对 $|I|$ 进行归纳法来证明。为了使归纳法成立，我们必须将该命题泛化为：

> 假设 $\Sigma = \Sigma_{i \in I}$ 且 $def_{i \in I} \Rightarrow \Sigma$ 且 $\Sigma'\theta \vdash def_i$ 成立 对所有 $i \in I$ 和某个满足 $\Sigma \subseteq \Sigma'$ 的 Σ' 均成立。那么，对于所有 $i \in I$ 和某个 C_i，$def_i \Rightarrow_{A_i} \langle C_i;\ \Sigma_i \rangle$ 。此外，$\Sigma' \cdot \emptyset \vdash \&_{i \in I} C_i \quad_A d$，并且存在满足 $\theta_\circ \cup \theta' \Vdash d$ 的 θ'，且 定义域$(\theta') = (A_\circ \cup_\circ \bigcup_{i \in I} A_i) \setminus \mathrm{ftv}(\Sigma)$。

含有 $|I| = 0$ 的基本情况显然成立。在归纳步骤中，假设 $|I| = n$，并且泛化后的命题已对 $n - 1$ 成立。接着，我们使用引理 B.18 证明 n。繁琐但简单的推理过程从略。 □

定义 *B.20*（类型方案与方案环境之间的关系）。

- 若对于所有 $t_2 \in \mathrm{inst}(\sigma_2)$，都存在 $t_1 \in \mathrm{inst}(\sigma_1)$使得 $t_1 \leq t_2$，则称类型方案 σ_1 比类型方案 σ_2更一般，记作 $\sigma_1 \sqsubseteq \sigma_2$。
- 若$\mathrm{dom}(\Sigma_1) = \mathrm{dom}(\Sigma_2)$，且对于所有 $x \in \mathrm{dom}(\Sigma_1)$，都有 $\Sigma_1(x) \sqsubseteq \Sigma_2(x)$，则称方案环境 Σ_1 比方案环境 Σ_2更一般，记作 $\Sigma_1 \sqsubseteq \Sigma_2$。

性质 B.21（tally* 的性质）。

(i) 若 $\theta \in tally^\star(c)$，则 $\theta \Vdash c$。*(ii)* 若 $\theta' \Vdash c$，则存在 $\theta \in tally^\star(c)$以及 θ''，使得对于所有 $\alpha \notin ftv(\theta)$，都有 $\alpha\theta' \simeq \alpha\theta\theta''$。*(iii)* 若 $\theta \in tally^\star(c)$，则 $dom(\theta) = ftv(c)$，且 $ftv(\theta)$ 是新鲜的。

证明。参见第 39] 页的 Castagna 等人 [13, 定理 A.35，以及 Castagna 等人 [10]。将其推广到带析取的约束上是直接的。

引理 B.22。如果 $\theta \Vdash c$ 且 $\alpha \in$ 定义域(θ)，那么存在 $\theta' \in \mathrm{tally}^\star(c)$ 使得$\mathrm{gen}(\alpha\theta') \sqsubseteq \mathrm{gen}(\alpha\theta)$。

证明。根据性质 B.21，我们得到 $\theta' \in \mathrm{tally}^\star(c)$，其中 $\mathrm{ftv}(\theta')$ 是新鲜的（因此 $\alpha \notin \mathrm{ftv}(\theta')$），并且存在某个$\theta''$ 使得

$$\alpha\theta \simeq \alpha\theta'\theta'' \tag{1}$$

假设 $A = \mathsf{ftv}(\alpha\theta)$。则 $\mathsf{gen}(\alpha\theta) = \forall A.\alpha\theta$。不失一般性， A 是新鲜的，且 $A \cap \mathsf{dom}(\theta'') = \emptyset$。假设 $u \in \mathsf{inst}(\mathsf{gen}(\alpha\theta))$。因此，存在 $\theta^\star$，满足 $\mathsf{dom}(\theta^\star) = A$ 且 $u = \alpha\theta\theta^\star$。于是

$$u = \alpha\theta\theta^\star \overset{(1)}{\simeq} \alpha\theta'\theta''\theta^\star \overset{A \cap \mathsf{dom}(\theta'') = \emptyset}{=} \alpha\theta'(\theta'' \dot\cup \theta^\star)$$

因此， $u \in \mathsf{inst}(\mathsf{gen}(\alpha\theta'))$，所以 $\mathsf{gen}(\alpha\theta') \sqsubseteq \mathsf{gen}(\alpha\theta)$。 □

引理 B.23（更一般环境下的约束重写）。假设$\Sigma \cdot \Gamma \vdash C \quad d$ 且 $\Sigma' \sqsubseteq \Sigma$，其中 Σ'。那么 $\Sigma' \cdot \Gamma \vdash C \quad_A d'$，其中 d'。此外，若 $\theta \Vdash d$。其中 θ，则 $\theta \cup \theta' \Vdash d'$，其中 θ'，且 定义域$(\theta') = A$。

证明。对 $\Sigma \cdot \Gamma \vdash C \quad d$ 的推导进行归纳法。分类讨论推导的最后一条规则。

- 情形 规则 RC-AND：由归纳假设可得。
- 情形 规则 RC-SUBTY：显然。
- 情形规则 RC-VAR：显然，因为 $\Sigma' \sqsubseteq \Sigma$ 意味着 定义域(Σ') = 定义域(Σ)。
- 情形 规则 RC-VAR-POLY：对该规则应用反演可得：

$$x \notin \mathsf{dom}(\Gamma)$$
$$\Sigma(x) = \forall B.t$$
$$C = x \mathrel{\dot\leq} t'$$
$$d = t \mathrel{\dot\leq} t'$$

我们有 $\Sigma' \sqsubseteq \Sigma$，因此 $\Sigma'(x) = \forall A.u$。然后根据规则 rc-var-poly

$$\Sigma' \cdot \Gamma \vdash C \rightsquigarrow_A \underbrace{u \mathrel{\dot\leq} t'}_{=:d'}$$

现在假设 $\theta \Vdash d$。那么

$$t\theta \leq t'\theta \tag{1}$$

注意到 $t \in \mathsf{inst}(\Sigma(x))$，因此结合 $\Sigma' \sqsubseteq \Sigma$，我们有某个 $\theta^\star$，使得 定义域$(\theta^\star) = A$ 且

$$u\theta^\star \leq t \tag{2}$$

假设 $A = \{\alpha_i \mid i \in I\}$，且不失一般性， $A \cap$定义域$(\theta) = \emptyset$。令 $\theta^\star = [t_i/\alpha_i \mid i \in I]$，其中 t_i。定义 $\theta' = [t_i\theta/\alpha_i \mid i \in I]$。那么 定义域$(\theta') = A$ 且 定义域$(\theta) \cap$定义域$(\theta') = \emptyset$, 并且我们有

$$u(\theta \dot\cup \theta') = u\theta^\star\theta \overset{(2) \text{ and Property A.3}}{\leq} t\theta \overset{(1)}{\leq} t'\theta \overset{A \text{ fresh}}{=} t'(\theta \dot\cup \theta')$$

。因此， $\theta \cup \theta' \Vdash d'$。

- 情况规则 RC-DEF：由归纳假设直接得到
- 分类规则 rc-分类：反演该规则可得：

$$\frac{\Sigma \cdot \Gamma \vdash C' \rightsquigarrow c' \qquad (\forall i \in I)\ \Sigma \cdot \Gamma \vdash \hat{C}_i \rightsquigarrow \hat{c_i} \qquad \Sigma \cdot \Gamma \twoheadrightarrow \Gamma_i \vdash C_i \rightsquigarrow c_i}{\Sigma \cdot \Gamma \vdash \underbrace{\mathtt{case}\ C'\ \mathtt{of}\ (\Gamma_i\ \mathtt{in}\ C_i\ \mathtt{unless}\ \hat{C}_i)_{i \in I}}_{=C} \rightsquigarrow \underbrace{c'\ \&\ \&_{i \in I}(\hat{c_i} + c_i)}_{=d}}$$

由归纳假设可得：

$$\Sigma' \cdot \Gamma \vdash C' \rightsquigarrow_B d''$$
$$\Sigma' \cdot \Gamma \vdash \hat{C}_i \rightsquigarrow_{\hat{B}_i} \hat{d}_i$$
$$\Sigma' \cdot \Gamma \mathbin{\rotatebox{90}{$\ll$}} \Gamma_i \vdash C_i \rightsquigarrow_{B_i} d_i$$

于是根据规则 rc-case

$$\Sigma' \cdot \Gamma \vdash C \rightsquigarrow_A d'$$
$$d' = d'' \,\&\, \&_{i \in I}(\hat{d}_i + d_i)$$
$$A = B \mathbin{\dot{\cup}} \dot{\bigcup}_{i \in I}(B_i \mathbin{\dot{\cup}} \hat{B}_i)$$

如果我们还假设 $\theta \Vdash d$，那么 $\theta \Vdash c'$ 且对于所有 $i \in I$，要么 $\theta \Vdash \hat{}c_i$，要么 $\theta \Vdash c_i$。于是根据归纳假设，对于某个 $\theta^\star$，我们还有

$$\theta \mathbin{\dot{\cup}} \theta^\star \Vdash d''$$
$$\mathsf{dom}(\theta^\star) = B$$

对于 $i \in I$，根据归纳假设，对于某个 θ_i，要么 $\theta_i \Vdash \hat{}d_i$，其中 $\mathsf{dom}(\theta_i) = \hat{}B_i$；要么$\theta_i \Vdash d_i$，其中 $\mathsf{dom}(\theta_i) = B_i$。不失一般性，我们可以扩展 θ_i 的定义域，使得 $\mathsf{dom}(\theta_i) = B_i \circ \cup \hat{}B_i$。定义 $\theta' = \theta^\star \circ \cup \circ \bigcup_{i \in I} \theta_i$。那么 $\mathsf{dom}(\theta') = A$ 且 $\theta \circ \cup \theta' \Vdash d'$，如所需。

结束分类讨论。 □

引理 B.24. 若 $\text{定义域}(\theta) \cap \text{自由类型变量}(\theta) = \emptyset$ 则 $\theta \Vdash \text{等价}(\theta)$。

证明。 假设 $\alpha \in \text{定义域}(\theta)$。由 $\text{定义域}(\theta) \cap \text{自由类型变量}(\theta) = \emptyset$ 可得 $\alpha\theta\theta = \alpha\theta$，由此可直接推出$\theta \Vdash \text{等价}(\theta)$。□

引理 B.25（约束生成与重写中的自由类型变量）。

(i) 若 $\Sigma \cdot \Gamma \mathbin{\rotatebox{90}{$\ll$}} C \rightsquigarrow_A d$，则 $ftv(d) \subseteq ftv(\Sigma) \cup ftv(\Gamma) \cup A \cup ftv(C)$ *(ii)* 若 $e : t \Rightarrow_A C$，则 $ftv(C) \subseteq A \cup ftv(t)$。*(iii)* 若定义 $\Rightarrow_A \langle C; \Sigma \rangle$，则 $ftv(C) \subseteq A$。

证明。 (i) 和 (ii) 的证明分别通过对相应推导进行归纳得出。(iii) 可由 (ii) 推出。

We n现在重述并证明第3.5节中的算法化类型推导完备性定理。

定理 *3.7*（程序算法类型判定的完备性）。给定程序$prog$和某个类型 t。若 $\vdash prog : t$，则存在程序约束 P、简单约束 d和类型替换 θ，使得 $prog : t \Rightarrow P$ 且 $\vdash P \rightsquigarrow d$ 且 $\theta \Vdash d$。

证明。 令 $prog = \text{letrec}\ \text{定义}_{i \in I}\ \text{in}\ e$。通过逆转规则 D-PROG，得到：

$$(\forall i \in I)\ \mathsf{envs}(def_i) = \langle \Sigma_i; \Sigma'_i \rangle \tag{1}$$
$$\Sigma_{i \in I} \vdash def_i\ \mathsf{ok} \tag{2}$$
$$\Sigma'_{i \in I} \cdot \emptyset \vdash e : t \tag{3}$$

定义 $\Sigma = \Sigma_{i \in I}$ 和 $\Sigma' = \Sigma'_{i \in I}$。此外，假设 $\Sigma'' = \Sigma''_{i \in I}$ 使得

$$def_{i \in I} \Rightarrow \Sigma''_i \quad (\forall i \in I) \tag{4}$$

注意，对于 定义$_i = (x: \quad \sigma = e)$前提 $\mathsf{ftv}(\sigma) = \emptyset$是必需的 定义 $_{i\in I} \Rightarrow \Sigma_i''$可由（1）通过应用逆转规则 D-DEF-ANNOT推出。

不失一般性，我们可以假设 $n = |I|$

$$
\begin{aligned}
def_{i\in I} &= (x_1 = e_1), \quad \ldots, (x_k = e_k), \quad (x_{k+1} : \sigma_{k+1} = e_{k+1}), \ldots, (x_n : \sigma_n = e_n) \\
\Sigma'' &= \{x_1 = \alpha_1, \quad \ldots, x_k = \alpha_k, \quad x_{k+1} = \sigma_{k+1}, \quad \ldots, x_n = \sigma_n\} \\
\Sigma &= \{x_1 = t_1, \quad \ldots, x_k = t_k, \quad x_{k+1} = \sigma_{k+1}, \quad \ldots, x_n = \sigma_n\} \\
\Sigma' &= \{x_1 = \forall A_1.t_1, \ldots, x_k = \forall A_k.t_k, \quad x_{k+1} = \sigma_{k+1}, \quad \ldots, x_n = \sigma_n\}
\end{aligned}
$$

对于 $\sigma_i = \forall A_i\circ\ t_i$对于 $i = k+1,\ \ldots\ldots,\ n$。根据（1），通过逆转规则 D-ENVS-NO-ANNOT 和 D-ENVS-ANNOT 可知，$A_i = \mathsf{ftv}(t_i)$对于 $i = 1,\ \ldots\ldots,\ n$。定义 $\tilde{\theta}: = [t_1/\alpha_1,\ \ldots\ldots,\ t_k/\alpha_k]$。于是$\Sigma = \Sigma''\theta$，因此根据（2）可得

$$\Sigma''\tilde{\theta} \vdash def_i\ \mathsf{ok} \quad (\forall i \in I)$$

根据（4）和引理B.19 可得

$$def_i \Rightarrow_{B_i} \langle C_i; \Sigma_i''\rangle \quad (\forall i \in I) \tag{5}$$

$$\Sigma'' \cdot \emptyset \vdash \&_{i\in I}\, C_i \rightsquigarrow_B \tilde{d} \tag{6}$$

$$\tilde{\theta} \dot\cup \theta^\star \Vdash \tilde{d} \tag{7}$$

$$\mathsf{dom}(\theta^\star) = B \dot\cup \dot\bigcup_{i\in I} B_i \setminus \mathsf{ftv}(\Sigma'') \tag{8}$$

对于某些 $\theta^\star$ 和 $B,\ B_i$ 是新鲜的。我们有 $\Sigma' \cdot \emptyset \vdash e: t$ 根据（3），因此根据引理 B.16

$$e : t \Rightarrow_{B'} C' \tag{9}$$

$$\Sigma' \cdot \emptyset \vdash C' \rightsquigarrow_{B''} d' \tag{10}$$

$$\theta' \Vdash d' \tag{11}$$

。对于某个 θ' 满足 定义域$(\theta') = B' \cup B''$ 和 $B',\ B''$ 是新鲜的。由（7）可知，根据性质 B.21和引理 B.22 存在某个 $\theta^\top$ 使得

$$\theta^\top \in \mathsf{tally}^\star(\tilde{d}) \tag{12}$$

$$\theta^\top \Vdash \tilde{d}$$

$$\mathsf{dom}(\theta^\top) \cap \mathsf{ftv}(\theta^\top) = \emptyset \tag{13}$$

$$\mathsf{gen}(\alpha_i\theta^\top) \sqsubseteq \mathsf{gen}(\alpha_i(\tilde{\theta} \dot\cup \theta^\star)) \overset{\mathsf{dom}(\theta^\star)\ \text{fresh}}{=} \mathsf{gen}(\alpha_i\tilde{\theta}) \quad \text{for all } i = 1, \ldots, k,$$

此外，$\Sigma' = \mathsf{gen}(\Sigma) = \mathsf{gen}(\Sigma''\tilde{\theta})$。为了证明 $\mathsf{gen}(\Sigma''\theta^\top) \sqsubseteq \Sigma'$，我们注意到两个环境具有相同的定义域 $\{x_1,\ \ldots\ldots,\ x_n\}$，并且对于$j \in \{k+1,\ \ldots\ldots,\ n\}$，有 $\mathsf{gen}(\Sigma''\theta^\top)(x_j) = \sigma_j = \Sigma'(x_j)$（注意，$\sigma_j$是封闭的），且对于 $i \in \{1,\ \ldots\ldots,\ k\}$，有 $\mathsf{gen}(\Sigma''\theta^\top)(x_i) = \mathsf{gen}(\alpha_i\theta^\top) \sqsubseteq \mathsf{gen}(\alpha_i\tilde{\theta}) = \Sigma'(x_i)$。

然后根据引理 B.23和（10）、（11）

$$\mathsf{gen}(\Sigma''\theta^\top) \cdot \emptyset \vdash C' \rightsquigarrow_{B'''} d'' \tag{14}$$

$$\theta' \dot\cup \theta'' \Vdash d'' \tag{15}$$

$$\mathsf{dom}(\theta'') = B''' \tag{16}$$

对于某个 θ''。根据（5）和（9）以及规则 C-PROG：

$$prog : t \Rightarrow P$$
$$P = \texttt{letrec}\,\langle \&_{i\in I}\, C_i ; \Sigma'' \rangle\ \texttt{in}\ C'$$

根据规则 RC-PROG、(6)、(12) 和 (14) 也

$$\vdash P \rightsquigarrow d$$
$$d = \mathsf{equiv}(\theta^{\top})\ \&\ d''$$

定义 $\theta = \theta^{\top} \circ \cup\, \theta' \circ \cup\, \theta''$。对于不相交性，我们推理如下：

- 定义域($\theta^{\top}$) 与 定义域($\theta_{\prime}$)不相交：由 (12) 和性质 B.21(iii)

$$\mathsf{dom}(\theta^{\top}) = \mathsf{ftv}(\tilde{d})$$

由 (6) 和引理B.25(i) 可得

$$\mathsf{ftv}(\tilde{d}) \subseteq \mathsf{ftv}(\Sigma'') \cup B \cup \bigcup_{i\in I} \mathsf{ftv}(C_i)$$

根据 (5) 和引理 B.25(iii)，$\mathsf{ftv}(C_i) \subseteq B_i$ 对所有 $i \in I$均成立。因此

$$\mathsf{dom}(\theta^{\top}) \subseteq \{\alpha_1, \ldots, \alpha_k\} \cup B \cup \bigcup_{i\in I} B_i$$

。注意到 定义域(θ') $= B' \cup B''$，其中 B'和 B''相对于 α_1、……、 α_k和 B 以及所有 B_i都是新鲜的，这证明了 定义域($\theta^{\top}$) ∩定义域(θ') $= \emptyset$。

- 定义域($\theta^{\top}$) 与 定义域($\theta_{\prime\prime}$) 不相交，因为 定义域($\theta_{\prime\prime}$) $= B_{\prime\prime\prime}$ 也相对于 α_1、……、 α_k和 B 以及所有 B_i都是新鲜的。
- 定义域($\theta_{\prime}$) 与 定义域($\theta_{\prime\prime}$) 不相交，这是因为（15）。

由（15）可得 $\theta \Vdash d''$，再由（13）和引理 B.24 可得 $\theta \Vdash$ 等价($\theta^{\top}$)。因此，$\theta \Vdash d$，如所需。□

项目	文件	模式匹配	分支	type 测试	对……进行类型测试 变量		对……进行类型测试 绑定变量	
					abs.	*%*	*abs.*	*%*
antidote	55	1635	2310	4	4	100.0	4	100.0
Etylizer	45	1792	3084	17	17	100.0	15	88.2
RabbitMQ	296	18886	31305	1759	1749	99.4	1719	98.3
Riak Core	104	5572	8034	268	268	100.0	263	98.1
标准库	93	18482	53841	3383	3358	99.3	3307	98.5
总计	**593**	**46367**	**98574**	**5431**	**5396**	**99.4**	**5308**	**98.4**

表3. 模式匹配中类型测试使用情况的实证研究。研究考察了以下项目版本：antidote 于 2022-09-14 的 Git 提交 1482b81；etylizer 于 2025-05-28 的 Git 提交 4822984；RabbitMQ 于 2025-07-02 的 Git 提交 ee652cb；Riak Core 于 2023-06-12 的 Git 提交 7b04e2c；标准库版本 27.1.1。

C 模式匹配的实证分析

为指导我们的设计选择，我们对六个具有代表性的 Erlang 项目中模式匹配里类型测试的使用情况进行了实证研究。结果如表 3所示。

总计，我们分析了 593 个源文件，其中包含 46,367 个模式匹配和 5,431 个类型测试。结果表明，类型测试几乎只应用于变量（99.4%），且主要应用于在模式本身中绑定的变量（98.4%）。因此，我们的类型系统侧重于为这类类型测试提供精确支持。

D 动态函数的实证分析

为了评估对完整依赖值的类型系统的实际需求，我们分析了六个 Erlang 项目语料库中用于动态访问元组的`erlang:element/2`、`erlang:setelement/3` 以及 lists 模块中的函数 12（例如 `lists:ukeysort/2`）。目标是确定索引参数有多大比例是静态已知的字面量常量，而非动态计算的值。我们的假设是，函数式代码中的大多数用法都使用常量索引，而类型覆盖结合单例整数类型能够有效处理这些情况。

表 4 中总结的结果支持这一假设。我们发现，`element/2`、`setelement/3`和列表模块函数总计被调用了 1820 次。其中绝大多数调用（1386 次，占 76%）使用整数字面量作为索引（例如，`element(2, Tuple)`）。这些情况适用于我们的重载类型方法，该方法能为每个常量索引提供精确的返回类型。其余 434 次调用（占 24%）则使用非常量表达式作为索引。

值得注意的是，几乎所有非常量索引的情况都来自标准库。这是意料之中的，因为标准库包含用于处理任意用户输入的通用实用函数。在分析中排除标准库后，其他五个项目中 743 次非常量索引调用有 645 次可以通过我们现有的覆盖进行验证。我们的类型系统能够涵盖的代码比例因此提高到 87%。

函数	字面量	非字面量
erlang:element	741	326
erlang:setelement	30	68
lists:keydelete	58	3
lists:keyfind	268	12
lists:keymap	0	1
lists:keymember	59	1
lists:keymerge	2	0
lists:keyreplace	28	4
lists:keysearch	59	3
lists:keysort	59	11
lists:keystore	33	0
lists:keytake	24	3
lists:ukeymerge	5	0
lists:ukeysort	20	2
总计	**1386**	**434**

项目	文件	字面量	非字面量
antidote	55	27	4
Etylizer	45	6	20
RabbitMQ	296	546	62
Riak Core	104	134	25
标准库	93	673	321
JSONE	4	0	2
总计	**597**	**1386**	**434**

表 4. 动态函数中依赖类型使用情况的实证研究。研究考察了以下项目版本：antidote 的 Git 提交 1482b81 （ 2022-09-14 ）； Etylizer 的 Git 提交 4822984 （ 2025-05-28 ）； RabbitMQ 的 Git 提交 ee652cb（2025-07-02）； Riak Core 的 Git 提交 7b04e2c（2023-06-12）； 标准库版本 27.1.1；JSONE 的 Git 提交 a1c481f（2024-28-11）。