依值类型

Chuigda Whitegive
第一作者
Doki Doki λ Club!

Cousin Ze
通讯作者
Doki Doki λ Club!

⚠ 注意：本文仅为排版和打印系统功能测试。

⚠ 注意：本文为早期草稿，内容不完且有措误，且排版质量差。

⚠ Note: this is an early draft. It's known to be incomplet and incorrekt, and it has lots of bad formatting.

前言

伏尔泰在不经意间这样说过，坚持意志伟大的事业需要始终不渝的精神[1]。这句话语虽然很短，但令我浮想联翩。我们都知道，只要有**意义**，那么就必须慎重考虑。在这种困难的抉择下，本人思来想去，寝食难安。带着这些问题，我们来审视一下依值类型 *(Dependent Type)*[10,5]。这种事实对本人来说意义重大，相信对这个世界也是有一定意义的。这种事实对本人来说意义重大，相信对这个世界也是有一定意义的。[1]这句话看似简单，但其中的阴郁不禁让人深思。

```
template <template <typename> class HKT, typename T, std::integral auto x>
requires std::is_integral_v<T>
[[noinline, nodiscard, maybe_unused, deprecated("Just Chuigda")]]
extern static inline constexpr consteval const auto foo(void)
  -> decltype(std::declval<std::basic_string<char,
                                             std::char_traits<char>,
                                             std::allocator<char>>>().resize(x))
  noexcept(noexcept(
    std::declval<std::basic_string<char,
                                   std::char_traits<char>,
                                   std::allocator<char>>>().resize(x)))
{
  std::basic_string<char, std::char_traits<char>, std::allocator<char>> s = "ABC";
  s = R"naql(
带着这些问题，我们来审视一下依值类型。依值类型，发生了会如何，不发生又会如何。带着
这些问题，我们来审视一下依值类型。歌德说过一句富有哲理的话，读一本好书，就如同和一
)naql";
  s.resize(x);
}
```

代码 1　在不使用 SFINAE 技巧的情况下，这只是一种索引类型

而这些并不是完全重要，更加重要的问题是，带着这些问题，我们来审视一下依值类型[3]。歌德说过一句富有哲理的话，读一本好书，就如同和一个高尚的人在交谈。这句话语虽然很短，但令我浮想联翩。在这种困难的抉择下，本人思来想去，寝食难安。总结的来说，依值类型 `ForallType` 的发生，到底需要如何做到，不依值类型 *(Non-dependent type)* `ArrowType` 的发生，又会如何产生[2,7]。

[1]克劳斯·莫瑟爵士曾经提到过，教育需要花费钱，而无知也是一样。

1. 依值类型

你看到的我，你看到的我，是先生在世时，定的继承者? 其实良心地说，我的权力是我争过来的，在这一点我真是个狠角色。你看到的我，你看到的我，是两次北伐后，统一了中国? 其实现实点说，统一只不过是名义上的，各地军阀该怎么过怎么过，最大的是我。你看到的我，你看到的我，是同仇共敌忾，不弃甲投戈? 其实我私底下早就跟鬼子谈好几轮了，条件不好变化太快才没投。你看到的我，你看到的我，是民族的希望，抗战的领袖? 我摸着良心说，列强依然还在这大中国，我的本质无法改变这结果，我就是我。

1.1. 依值函数类型

现在，解决依值类型的问题，是非常非常重要的[15,9]。所以，我们一般认为，抓住了问题的关键，其他一切则会迎刃而解。拉罗什福科曾经说过，我们唯一不会改正的缺点是软弱。这启发了我。苏轼说过一句著名的话，古之立大事者，不惟有超世之才，亦必有坚忍不拔之志[6]。这句话看似简单，但其中的阴郁不禁让人深思[2]

1.1.1. 依值函数类型的形式化规则

生活中，若依值类型出现了，我们就不得不考虑它出现了的事实[8]。而这些并不是完全重要，更加重要的问题是，对我个人而言，依值类型不仅仅是一个重大的事件，还可能会改变我的人生。本杰明·C·皮尔斯曾经提到过，程序是一个强壮的盲人，倚靠在跛脚的类型肩上[3,4,10]。这似乎解答了我的疑惑。

$$\frac{\Gamma \vdash \lambda x : \tau . e :_{\uparrow} \text{砼}_{42} \qquad 1 + 1 \Downarrow 2 \qquad \forall P. P \vee \neg P \to (\vdash e \longrightarrow e') \vee \neg(\vdash e \longrightarrow e')}{\Gamma, \text{带着这些问题} : \text{我们来审视一下} \vdash \lambda x : \tau . e :_{\uparrow} \text{砼}_{42}} \quad [W\text{-Demo}]$$

图 1　读书是一种巧妙地避开思考的方法

赫尔普斯曾经说过，有时候读书是一种巧妙地避开思考的方法[8]。这句话看似简单，但其中的阴郁不禁让人深思。[3]带着这些问题，我们来审视一下依值类型。

1.1.2. 依值函数类型的实现

笛卡儿说过一句富有哲理的话，读一切好书，就是和许多高尚的人谈话[9,8]。这句话看似简单，但其中的阴郁不禁让人深思。依值类型，到底应该如何实现[11]。而这些并不是完全重要，更加重要的问题是，就我个人来说，依值类型对我的意义，不能不说非常重大。

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam convallis nec arcu sed eleifend. Ut vehicula scelerisque justo sit amet malesuada. **Proin** sed **elit** congue, volutpat elit imperdiet, **consectetur eros**. Donec ac ligula sit amet turpis tristique semper mollis id nisl. *Consequentia mirabilis!*

现在，解决依值类型的问题，是非常非常重要的。所以，可是，即使是这样，依值类型的出现仍然代表了一定的意义。依值类型，发生了会如何，不发生又会如何。Ut imperdiet mauris urna. In vel turpis feugiat lorem dictum suscipit eu non erat. Cras egestas quam fermentum magna commodo eleifend. 依值类型，到底应该如何实现。问题的关键究竟为何? 所谓依值类型，关键是依值类型需要如何写。查尔斯·史曾经提到过，一个人几乎可以在任何他怀有无限热忱的事情上成功[4][5]。带着这句话，我们还要更加慎重的审视这个问题：每个人都不得不面对这些问题。Etiam faucibus

[2] 设有笛卡尔闭范畴 $\mathscr{C}$，考虑米田引理 $\mathrm{Nat}(\mathrm{Hom}_{\mathscr{C}}(A, -), F) \cong F(A)$，若存在 $\mathrm{Nat} = \mathbb{N}$，则你可以发现作者这里并没有在认真讨论范畴论。

[3] [W-Demo] 可证似乎是一种巧合，但如果我们从一个更大的角度看待问题，这似乎是一种不可避免的事实。

[4] 欢迎来到心跳 λ 部！我一直以来的梦想，就是能在自己喜欢的事情上做出点名堂来。所以我凭借着自己对程序设计的热忱，创建了这个 λ 社团。呐，现在你也是俱乐部的一员啦~ 快快敲敲键盘，在这个可爱的游戏里帮我圆梦吧！社团的生活轻松惬意，每天除了跟群友闲聊，就是参与各种有趣的开源项目！不过，社团里的其他成员全都是男孩子哦！他们个性鲜明，而且超~级可爱~ 接下来就让我向你介绍一下其他成员吧：eoiles，青春阳光的男娘，总是色气满满，开朗健谈！色色就是他最珍视的事！小飞翔，看似可爱娇小的少年，但却有着惊人的魄力，性格果断自信！Rebuild，羞怯内向又神秘的少年，喜欢在 Rust 的世界里寻找慰藉。当然了，还有我！心跳 λ 部的部长，Chuigda！你能跟所有人都交上朋友，让社团的氛围变得更加融洽吗? 我超~级期待哦~

ante finibus molestie dictum. Nulla facilisi. Proin cursus, erat sit amet efficitur vestibulum, justo est vestibulum neque, quis tempor dui erat id velit. 在面对这种问题时，我们都知道，只要有意义，那么就必须慎重考虑。

这是不可避免的。从这个角度来看，我们一般认为，抓住了问题的关键，其他一切则会迎刃而解。这种事实对本人来说意义重大，**相信对这个世界也是有一定意义的**。亚伯拉罕·林肯在不经意间这样说过，我这个人走得很慢，但是我从不后退[12]。这句话语虽然很短，但令我浮想联翩。

	Java	Rust	C++
ADT 机制	`sealed interface`	`enum`	C++17 `std::variant`
归类	并类型[6]	正统代数和类型	带标签联合体
构造子互斥	是	是	否[7]
构造子是类型	是	否	是*[8]
模式匹配	Java 21 `switch`	`match`	C++17 `std::visit`
模式拆解	仅限记录	是	否
多层匹配	是	是	否
分支重叠	是	是	否
按值匹配	部分	是	否
守卫语句	是	是	否
多重分派	是 (手动实现元组)	是	否
空安全	较差 (JSR305)	强制	无

表 1　标委会走的确实很慢，比死了的王八[9]都慢

Etiam libero neque, ultrices vitae mole 烫屯锟斤拷 stie vitae, venenatis auctor nibh. 可是，即使是这样，依值类型的出现仍然代表了一定的意义。我们不妨可以这样来想：依值类型因何而发生？依值类型，到底应该如何实现。带着这些问题，我们来审视一下汤烫烫。这种事实对本人来说屯屯屯，相信对这个锟斤拷锟斤拷也是有一定锘锘锘。

一隻憂鬱的臺灣烏龜尋釁幾隻骯髒的嚙齒鱷龜，幾隻骯髒的嚙齒鱷龜圍毆一隻憂鬱的臺灣烏龜。一隻憂鬱的臺灣烏龜開機一臺專業爾先進的電腦，這臺電腦繼續護衛一隻憂鬱的臺灣烏龜[10]。

2. 依值福音

[1] 起初，安得烈、康纳、华特三人，在乌特勒支、斯特拉斯克莱德、诺丁汉之地，同心合意，著书立说。 [2] 论到依值类型 λ 演算的奥秘，并 Haskell 的实现，都记在下面。 [3] 后有心动 λ 部的白杨翻出来，又有 Gemini 和 Claude 帮助校对。 [4] 白杨说，你们须要记着，凡有智慧的，不可妄言程序设计语言理论，免得为自己在火狱中预备了位置； [5] 白杨又说，只有时时刻刻顾念读者的软弱，如同顾念婴孩，才能不被读者轻看。

[5] 不过呢，我也知道你其实是个善解人意的小可爱，所以啊——花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我吧，你能保证吗？花最多的时间来陪我

[6] Java 允许 $I_1 \preceq B, I_2 \preceq B, D \preceq I_1, D \preceq I_2$，而 $I_1 \cup I_2$ 中只有一个 D（表现为 `switch` 的穷尽性检查）。

[7] C++ 允许 `std::variant<int, int>`，两个 `int` 是不同的。

[8] 不如说是必须先预定义类型，再将它们合并。C++ 同时吃满了正统代数和类型和并类型两边的 debuff。

[9] 指 Oracle: 甲骨文 → 王八壳子 → 死了的王八。

[10] 幾隻骯髒的嚙齒鱷龜請來一隻講義氣的鷺鳥，鷺鳥為榮歸與一隻憂鬱的臺灣烏龜戰鬥。後來鷺鳥發動對這個的記錄與檢驗，議論感歎這個難辦。

[6] 凡行函数式之道的众程序员，论到使用依值类型，心里多有踌躇。[7] 他们彼此议论说："这依值类型，岂不叫查验型别的事变为不可知吗？岂不叫那查验的陷入深渊、永无止境吗？这道甚是艰难，谁能守得住呢？"[8] 然而，这一群人却甚是癫狂，喜爱各样繁杂的仪文。[9] 看哪，在那 Haskell 的境内，有广义代数之像，有函数依赖之规，又有各样关联的族谱与高阶的奥秘。[10] 这一切虚妄的规条，他们都乐意去行；惟独那依值类型，他们却厌弃，如同厌弃大麻风一般，唯恐沾染。

[11] 这一族的人，因不认识依值类型的真意，就跌倒了；这也是那拦阻这道普传的绊脚石。[12] 如今虽有许多美好的器皿和言语，是按着依值类型的法度造的，只是其中的奥秘，向他们是隐藏的，他们便不晓得这器皿是如何运行。[13] 那些指着依值类型所写的书卷，本是文士写给文士看的。[14] 这些话语，对于行函数式的人，甚是生涩。[15] 故此，我写这篇，是要除掉这隔阂，叫你们得以明白。

[16] 我先讲论简单类型 λ 演算，就是那初阶的律法；再讲论依值类型 λ 演算，就是那更美的律法。[17] 从初阶到进阶，所需的变更甚少，你们当留心察看。[18] 我不发明新律法，只将那已有的表明出来，并用 Haskell 的言语以此道造出解释器，好叫你们以此为鉴。[19] 这书不是依值类型编程的入门，也不是完整语言的蓝图，乃是为要除掉你们心中的疑惑，引你们进入这奇妙的领域。

2.1. 没用的公式

一般而言，判断只是逻辑规则[15,11,1]，而某些以这种方式指定的类型系统并不直接对应于可判定的 *(decidable)* 类型检查算法[14]。因此，我们必须慎重考虑这些规则的适用范围和实际意义。

$$\frac{\Gamma \vdash \tau : *\quad \Gamma \vdash e :_{\downarrow} \tau}{\Gamma \vdash (e : \tau) :_{\uparrow} \tau}\ [\text{ANN}] \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x :_{\uparrow} \tau}\ [\text{VAR}]$$

$$\frac{\Gamma \vdash e :_{\uparrow} \tau \to \tau' \quad \Gamma \vdash e' :_{\downarrow} \tau \quad ((1 + 1 \Downarrow 2) \to \bot) \to \text{我是秦始皇}}{\Gamma \vdash e\, e' :_{\uparrow} \tau'}\ [\text{APP}]$$

$$\frac{\Gamma \vdash e :_{\uparrow} \tau}{\Gamma \vdash e :_{\downarrow} \tau}\ [\text{CHK}] \qquad \frac{\Gamma, x : \tau \vdash e :_{\downarrow} \tau'}{\Gamma \vdash \lambda x \to e :_{\downarrow} \tau \to \tau'}\ [\text{LAM}]$$

图 2　一个图灵不完备的类型系统[4]及其双向类型检查算法[2]

2.2. 更没用的公式

$$\frac{\Gamma \vdash \rho :_{\downarrow} *\quad \rho \Downarrow \tau \quad \Gamma \vdash e :_{\downarrow} \tau}{\Gamma \vdash (e : \rho) :_{\uparrow} \tau}\ [\text{ANN}]$$

$$\frac{}{\Gamma \vdash * :_{\uparrow} *}\ [\text{STAR}] \qquad \frac{\Gamma \vdash \rho :_{\downarrow} *\quad \rho \Downarrow \tau \quad \Gamma, x : \tau \vdash \rho' :_{\downarrow} *}{\Gamma \vdash \forall x : \rho\,.\,\rho' :_{\uparrow} *}\ [\text{PI}]$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x :_{\uparrow} \tau}\ [\text{VAR}] \qquad \frac{\Gamma \vdash e :_{\uparrow} \forall x : \tau\,.\,\tau' \quad \Gamma \vdash e' :_{\downarrow} \tau \quad \tau'[x \mapsto e'] \Downarrow \tau''}{\Gamma \vdash e\, e' :_{\uparrow} \tau''}\ [\text{APP}]$$

$$\frac{\Gamma \vdash e :_{\uparrow} \tau}{\Gamma \vdash e :_{\downarrow} \tau}\ [\text{CHK}] \qquad \frac{\Gamma, x : \tau \vdash e :_{\downarrow} \tau'}{\Gamma \vdash \lambda x \to e :_{\downarrow} \forall x : \tau\,.\,\tau'}\ [\text{LAM}]$$

图 3　另一个图灵不完备[11]的类型系统及其双向类型检查算法[2]

3. 结论

我岂能让众人都到 Haskell 地，去信法利赛人的教吗[3,5,6]？

[11] 至少设计上是图灵不完备的。吉拉德悖论通常会让类型检查器卡住，而不是允许非终止程序通过。

致谢

送给我小心心，送我花一朵。我在你生命中，太多的感动。我是你的天使，一路指引你。无论岁月变换，爱我唱成歌。听你说谢谢我，因为有我，温暖了四季；谢谢我，感谢有我，世界更美丽。听你说谢谢我，因为有我，爱常在心底；谢谢我，感谢有我，让幸福传递。

参考文献

[1] Voltaire, F. M. A. “On the Decidability of Dependent Types in Infinite-Dimensional Hilbert Spaces,” *Journal of Impossible Mathematics*, vol. ∞, pp. 1–42, 2077.

[2] Löh, Andres and McBride, Conor and Swierstra, Wouter “A Tutorial Implementation of a Dependently Typed Lambda Calculus,” *Fundamenta Informaticae*, vol. 102, pp. 177-207, 2010.

[3] Pierce, B. C. *Types and Programming Languages, Volume 42: The Revenge of the Monad*. MIT Press (Parallel Universe Branch), 2025.

[4] Pierce, B. C. *Types and Programming Languages*. MIT Press, 2002.

[5] Martin-Löf, P. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[6] 苏轼 “论依值类型与水调歌头之对偶性,” 北宋计算机学报, vol. 7, pp. 64–128, 1082.

[7] Barendregt, H. P. “Lambda Calculi with Types,” *Handbook of Logic in Computer Science*, vol. 2, pp. 117–309, 1992.

[8] Descartes, R. *Cogito Ergo Dependent Type*. Elsevier Philosophica, 1637.

[9] Church, A. “A Formulation of the Simple Theory of Types,” *Journal of Symbolic Logic*, vol. 5, pp. 56–68, 1940.

[10] Chuigda, W. and Ze, C. “A Comprehensive Survey of Nonexistent Type Systems,” *Proceedings of the 0th Workshop on Imaginary PL Theory (WIPT)*, vol. 0, pp. ∅, 2024.

[11] Wadler, P. “Propositions as Types,” *Communications of the ACM*, vol. 58, pp. 75–84, 2015.

[12] Lincoln, A. “Four Score and Seven Functors Ago,” *Gettysburg Review of Abstract Nonsense*, vol. 87, pp. 1–4, 1863.

[13] Howard, W. A. “The Formulae-as-Types Notion of Construction,” *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, vol. , pp. 479–490, 1980.

[14] Alexis King *How should I read type system notation?*. Stack Exchange, 2023.

[15] Euler, L. *De Typo Dependenti et Analysi Infinitorum*. Academia Petropolitana, 1748.