我和权威的故事

每个人小时候心里都是没有权威的，就像每个人小时候也都不相信广告一样。可是权威就像广告，它埋伏在你的潜意识里。听一遍不信，听两遍不信，……，直到一千遍的时候，它忽然开始起作用了，而且这作用越来越强。

消灭广告所造成的幻觉，最好的办法就是去尝试，去实地的考察它。有些虚幻的东西只要你第一次尝试就会像肥皂泡一样破灭掉。可是如果你不主动去接触它，它就会一直在你脑海里造成一种美好神圣的假象。越是得不到的越是觉得美好。很神奇的一个现象就是，权威对人思想的作用其实也跟广告一样。

上大学以前的人因为没有专业，所以还不怎么崇拜权威，大不了追追歌星，影星，球星啥的。而进了大学之后，就会开始对本领域的权威耳濡目染。一遍，两遍，一千遍的听到同学们仰慕某“牛人”或者“大师”的名字，虽然从来没亲身见过，不知不觉就对这人产生了崇拜心理，然后自愧不如。不知不觉的，自己也开始附和这些说法，不自觉地提到这些大师的名字，引用他们说的话作为自己的行动指南。

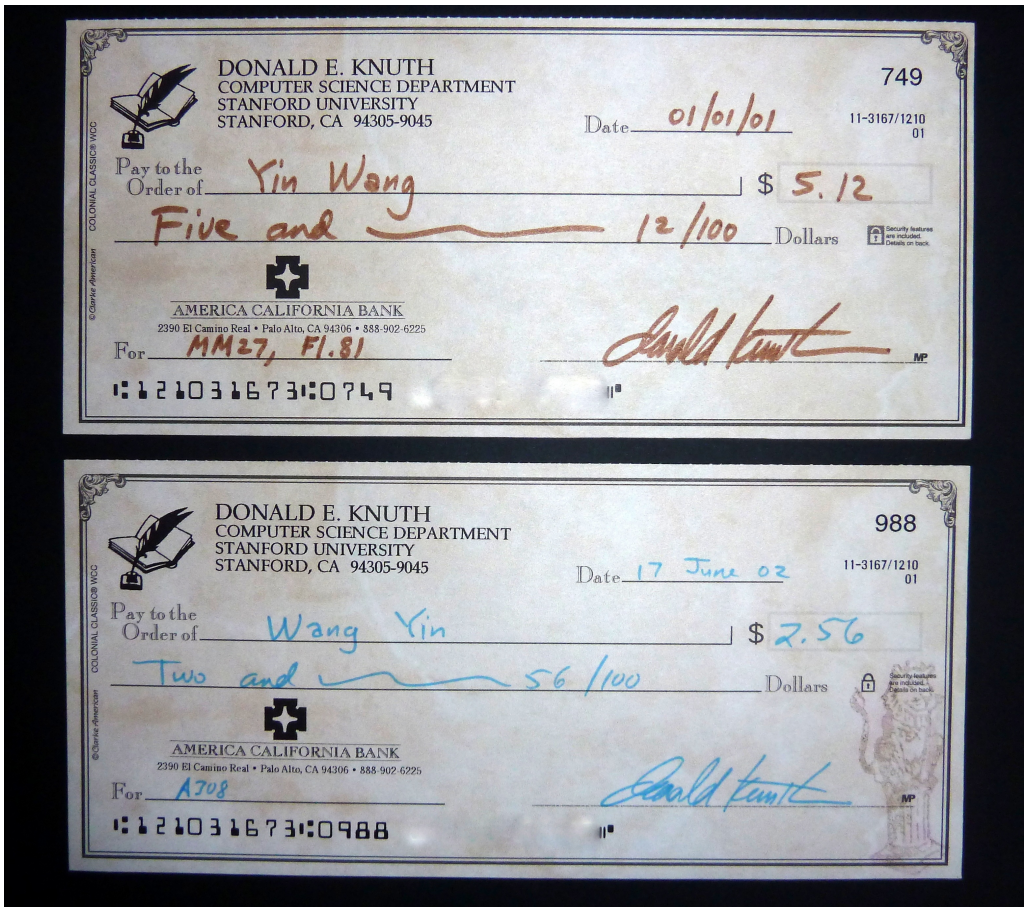
Donald Knuth, Dennis Ritchie, Ken Thompson, Rob Pike, … 就是通过这些途径成为了很多计算机学生的权威。以至于几十年以后，他们的一些历史遗留下来的糟糕设计和错误思想还被很多人奉为神圣。

Donald Knuth

很多人（包括我）都曾经对 Knuth 和他的 The Art of Computer Programming (TAOCP) 极度崇拜。在我大学和研究生的时候，有些同学花了不少钱买回精装的 TAOCP 全三卷，说是大概不会看，但要供在书架上，镇场子。当时我本着“书非借不能读也”的原则，再加上搬家的时候书是最费力气的东西，所以坚决不买书。我就从图书馆把 TAOCP 借了来。说实话我哪里看的下去啊？那里面的程序都是用一个叫 MIX 的处理器的汇编语言写的。一个字节只有6位，每位里面可以放一个十进制数（不是二进制）！还没开始写程序呢，就开始讲数学，然后就是几十页的公式推导，证明…… 接着我就睡着了。但我总是听说有人真的看完过 TAOCP，然后就成为了大师。比尔盖茨也宣称：“要是谁看完了 TAOCP，请把简历投给我！” 在这一系列的号召和鼓吹之下，我好几次的把 TAOCP 借回来，下定决心这次一定看完这旷世奇书。每次都是雄心勃勃的开始，可从来就没看完过开头那段 MIX 机器语言和数学公式。

看不懂 TAOCP 总是感觉很失败，因为看不懂 TAOCP 就成不了“大师”，可我仍然认为 Knuth 就是计算机科学的神，总能从他那学点什么吧，所以又开始折腾他的其他作品。这就是为什么我开始用 TeX，并且成为中国 TeX 界的主要“传教士”之一。为了 TeX，我把 Knuth 的 TeXbook 借回来，从头到尾看了两遍，做完所有的习题，包括最难最刁钻的那种“double bend”习题。接着又开始看 MetaFont Book，开始使用 MetaPost 进行绘图。开头还挺有成就感，可是不多久就发现学会的那些 TeX 技巧到了临场的时候就不知道该怎么用，然后就全都忘记了。这就是为什么我把 TeXbook 看了两遍，可是看完第二遍之后不久还是忘记得一干二净。

师兄师姐看到我用 TeX，说怎么折腾这么过时的玩意儿。我很气愤他们以及国内学术界居然都用 Word 排版论文，就开始针锋相对，写出一系列煽动文章鼓吹 TeX 的种种好处，打击“所见即所得”这种低智商玩意儿。这还不够，又开始折腾 Knuth 设计的 MMIX 处理器，并且认为 MMIX 的寄存器环就是世界上最先进的设计。发现一些无关紧要的小错，就给 Knuth 发 email，居然拿到两张传说中的“Knuth 支票”，并且一度引以为豪。当然像所有拿到 Knuth 支票的人一样，你是不会去兑现它的，甚至有人把它们像奖状一样放在相框里。我还没那么疯狂，那两张支票一直在它们原来的信封里。多年以后我到美国想兑现那支票的时候，发现它们已经过期了。



当你心里有了这样的权威，其他人的话你是不可能听得进去的，就算他们其实比你心目中的权威更具智慧也一样。在清华的时候我有时候去姚期智的小组听客串讲座。有一次请来了美国某大学一个教授讲算法，不知道怎么的我就跟他聊起 TAOCP，大概是想请教他如何学习算法。他跟我说 Knuth 的书已经比较过时了，你可以看看 MIT 的那本《算法导论》。可是这位教授的名气怎能和 Knuth 相比，这话我恁是没有听进去，仍然认为 TAOCP 隐藏了算法界最高的机要，永恒的珍宝。

在清华的时候我很喜欢一门叫做“计算几何”的课，就经常跟那门课的老师交流思想。有一次我在 email 里面提到 Donald Knuth 是我的偶像，那位老师很委婉的回复道：“有偶像很好啊，Knuth 也曾经是我的偶像。”我对“曾经”这两个字感到惊讶：难道这意味着 Knuth 现在不是他的偶像了？在我执意的询问之下他才告诉我，其实世界上还有很多更聪明的人，Knuth 并不是计算机科学的一切。你应该多看看其他人的作品，特别是一些数学家的。然后他给了我几个他觉得不错的人的名字。

现在回想起来，这些话对我是有深远作用的。那位老师虽然在系里的“牛人”们眼里是个研究能力（也就是发 paper 能力）不强的人，但是他却对我的人生转折有着强有力的作用。他引导了我去追寻自己真正的兴趣，而不是去追寻虚无的名气。我发现很多人都在为着名气而进行一些自己其实不感兴趣的事情，去做一些别人觉得“牛气”的事情。我真希望他们遇到跟我一样的好老师。

在现在看来，Knuth 的 TAOCP 就是所谓的“神圣的白象”（white elephant）。大家都把它供起来，其实很少有人真的看过，却要显得好像看过一样，并且看得津津有味。这就让试图看懂它的人更加自卑和着急，甚至觉得自己智商有问题。别人都看过了，我怎么就看不懂呢？其实 TAOCP 里面的大部分算法都不是 Knuth 自己设计的，而且他对别人算法的解释经常把简单的问题搞得很复杂。再加上他执意要用汇编语言，又让程序的理解难度加倍。

有一句话说得好：“跟真正的大师学习，而不是跟他们的徒弟。”如果你真的要学一个算法，就应该直接去读那算法的发明者的论文，而不是转述过来的“二手知识”。二手的知识往往把发明者原来的动机和思路都给去掉了，只留下苍白无味，没有什么启发意义的“最后结果”。确实是这样的，多年以后当我看见 Knuth 计划中的几卷新的 TAOCP 的目录时，发现其中大部分的东西我已经通过更容易的方式学到了，因为我找到了这些知识的源头。

所以之前的那位访问清华的教授说的其实是实话，Knuth 真的落伍了，可是就算在美国也少有人知道或者承认这个情况。有一次看一个对世界上公认最厉害的一些程序员的采访，包括总所周知的一些大牛，以及 ML 的设计者 Robin Milner，Haskell 的设计者之一 Simon Peyton Jones 等人。也不知道采访者是什么心理，在对每个人的采访中他都问，你看过 TAOCP 吗？大部分人都说看过，真是了不起的巨著，很重要啊云云。只有 Robin Milner （如果我没记错的话）比较搞笑，他说我希望我看过，但是可惜实在没时间。我一直把 TAOCP 垫在我的显示器下面，这样我工作时就可以一直看着它们 :)

Knuth 说“premature optimization is the root of all evil”，然而他自己却是非常喜欢用 premature optimization 的人。他的代码里到处是莫名其妙的小聪明，小技巧。把代码弄得难懂，实际上却并没有得到很多性能

的提高。有一次看 MMIX 处理器的模拟程序，发现他用来计算一个寄存器里的“1”的个数的代码非常奇怪。本来写个循环，或者用那种从末位减 1 的做法就可以了，结果他的代码用了 Programming Pearls 里面一个古怪的技巧，费了我半天时间才看懂，后来我发现这个技巧其实还不如最简单的方法。就是这些细小却又蹊跷的做法，使得 Knuth 的代码用细节掩盖了全局，所以到最后我其实也没从大体上搞懂一个处理器的模拟器应该如何工作。直到后来到 Indiana 学习了程序语言的理论之后我才发现，其实处理器模拟器（以至于处理器本身）的工作原理很简单，因为它就是一个机器代码的解释器。使用跟高级语言解释器同样的结构，你可以比较容易的写出像 MMIX 模拟器那样的东西。

Knuth 最重要的一个贡献恐怕是程序语言的 parsing （语法分析），比如 LR parsing，然而 parsing 其实是一个基本不存在的人造问题。它的存在是因为人们的误解，以为程序语言需要有跟人类语言一样的语法，所以把程序语言搞得无端的复杂和困难。如果你把语法简化一下，其实根本用不着什么 LR，LALR。我最近给我自己设计的语言写了一个 parser ，从头到尾只花了两个小时，500 行 Java 代码，包括了从 lexer 一直到 AST 数据结构的一切。完全手写的代码，根本没用任何复杂的 parsing 技术和 YACC 之类的工具，甚至正则表达式都没有用。之所以可以这样，因为我的语法设计让 parsing 极其容易，比 Lisp 还要容易。Knuth 过度的强调了 parsing。他的误导使得很多人花了几十年时间来研究 parser，到现在还在不时地提出新的技术，用于设计更加复杂的语法。何必呢？这只会让程序员和编译器都更加痛苦。如果这些人把时间都花在真正的问题上，那今天的计算机科学不知道要美好多少。

几乎每一本编译器教材都花大量篇幅来讲述 DFA, NFA, lexing, LL, LR, LALR…… 几乎每个学校的编译器课程都会花至少 30% 的时间来做 parser，折腾 LEX，YACC 等工具，而对于编译器真正重要的东西却没有得到很多的训练。这就是为什么 Kent Dybvig 的编译器课程如此有效，因为 Scheme 的语法非常简单，我们根本没有花时间来做 parser。我们的时间用在了思考真正的问题：做优化，实现尾递归，高阶函数…… 很多语言梦寐以求却又做不好的东西。这样的课程给了我可以发挥自己潜力的余地，我的课程编译器里面具有大量的独创写法，我的 X64 机器代码生成器生成极其短小的代码，让 Kent Dybvig 都在背地里琢磨是怎么回事。这些东西到现在也许仍然是世界上最先进的技术。

一个人的思维方式似乎决定了他设计的所有东西。Knuth 的另一个最重要的发明，文学编程（Literate Programming）其实也是多此一举，制造麻烦。文学编程的错误在于认为程序语言应该像人类语言，应该适应所谓的“人类思维”。然而程序语言却是在很多方面高于人类语言的，它不应该受到人类语言里的糟粕的影响。把程序按照 Knuth 的方式分开在不同的文章段落里，造成了代码之间的关系很难搞清楚，而且极其容易出错。这个错误与“Unix 哲学”的错误类似，把程序作为一行一行的文本，而不是一个像电路图一样的数据结构。我不想在这里细说这个问题，对此我专门写了一篇文章，讲述为什么文学编程不是一个好主意。

TeX 其实也是异常糟糕的设计。它过度的复杂，很少有人搞得懂怎么配置。经常为了一个简单的效果折腾很久，然后不久就忘了当时怎么做的，回头来又得重新折腾。原因就是因为 TeX 的设计缺乏一致性，特殊情况太多，而且组合（compose）能力很差。所以你需要学太多东西，而不是跟象棋一样只需要学习几个非常简单的规则，然后把它们组合起来形成无穷的变化。

在程序语言设计者看来，TeX 的语言是世界上最恶劣的设计之一，但如果没有这个语言，它也许会更加糟糕。其实 TeX 之所以有一个“扩展语言”，有一个鲜为人知的小故事。在最早的时候 Knuth 的 TeX 设计里并没有一个语言。它之所以有一个语言是因为 Scheme 的发明者 Guy Steele。Knuth 设计 TeX 的那个时候 Steele 碰巧在斯坦福实习。他听说 Knuth 要设计一个排版系统，就建议他设计一个语言，以应付以后的扩展问题。在 Steele 的强烈建议和游说之下，Knuth 采纳了这个建议。可惜的是 Steele 并没能直接参加语言的设计，在短短的一个夏天之后就离开了斯坦福。

Knuth 的作品里面有他的贡献和价值，TeX 的排版算法（而不是语言）也许仍然是不错的东西。可是如果因为这些好东西爱屋及乌，而把他所推崇的那些乱七八糟的设计当成神圣的话，那你自己的设计就逃脱不出同样的思维模式，让简单的事情变得复杂。仍然对 TeX 顶礼膜拜的人应该看一下 TeXmacs，看看它的作者是如何默默无闻的，彻彻底底的超越了 TeX 和 Knuth。

在我看来，Knuth 是个典型的精英主义者，他觉得自己做的都是最好，最有“格调”的。他利用自己的权威和特立独行来让用户屈服于自己繁复的设计，而不是想法设计出更加易用的工具。TeX 的版本号每次更新都趋近于圆周率π，意思是完美，没有 bug。他奖励大额的支票给发现 TeX 代码里 bug 的人，用于显示自己对这些代码的自信，然而他却“冰封”了 TeX 的代码，不再填加任何新东西进去，也不再简化它的设计。当然了，如果不改进代码，自然就不会出现新的 bug，然而它的设计也就因此固步自封，停留在了几十年以前。更奇怪的是，“TeX”这个词居然不按照正常的英语发音逻辑读成”teks”。每当有人把它“读错”，就有“高手”打心眼里认为你是菜鸟，然后纠正：“那个词不读 teks，而要读‘特喝’，就像希腊语里的 chi，又像是苏格兰语的 loch，德语的 ach，西班牙语的 j 和俄语的 kh。”也许这就叫做附庸风雅吧，我是纯种的欧洲人！;-) 当一个软件连名字的发音都这么别扭，这么难掌握，那这个软件用起来会怎样？每当你提到 TeX 太不直观，就有人跟你说：“TeX 是所想即所得，比你的所见即所得好多了！”可事实是这样吗？看看 TeXmacs 吧，理解一下什么是“所见即所得+所想即所得”二位一体。

我跟 Knuth 的最后一次“联系”是在我就要离开清华的时候。我从 email 告诉他我觉得中国的研究环境太浮躁了，不是做学问的好地方，想求点建议。结果他回纸信说：“可我为什么看到中国学者做出那么多杰出的研究？计算机科学不是每个人都可以做的。如果你试了这么久还不行，那说明你注定不是干这行的料。”还好，我从来没有相信他的这段话，我下定了决心要证明这是错的。多年的努力还真没有白费，今天我可以放心的说，Knuth 你错了，因为我已经在你引以为豪的多个方面超过了你。

Unix

Unix 的创造者们是跟 Knuth 非常类似的权威，他们在我的心目中也曾经占据了重要的位置，以至于十年前我写了一篇文章叫《完全用 Linux 工作》，大力鼓吹 Unix 的“哲学”，甚至指出 Linux 不能做的事情就是不需要做的，并且介绍了一堆难用的 Unix 工具，引得很多人去折腾。可如果你知道我现在对 Unix 的态度，肯定会大吃一惊，因为在经过努力之后，我成功的“忘记”了 Unix 的几乎一切，以至于本科刚毕业的学生都会以为我是脑盲，并且以为可以在我面前炫耀自己知道的 Linux 技巧。他们不会明白，在我心里 Unix/Linux 的设计是计算机软件界目前面临的大部分问题的罪魁祸首，而他们显示给我看的，只不过是 Unix 的思想和精英主义给程序员造成的精神枷锁。其实我并不会忘记 Linux 的设计，但我已经下意识的以熟悉 Linux 的奇技淫巧为耻，所以很多时候我即使知道也要装作不知道。因为我是机器的主宰，而不是它的奴隶，所以我总是想办法让机器去帮我做更多的事，帮我记住那些无聊的细节，而不是去顺从它的设计者所谓的“哲学”。

评论 Unix 和它的后裔们总是一件尴尬的事情，因为你提到它们的任何一个缺点，都会被很多人认为是优点。GNU 的含义是“GNU is Not Unix”，但很可惜的是 GNU 和 Linux 的设计从来没有摆脱过 Unix 思想的束缚。Unix 的内存管理，进程，线程，shell，进程间通信，文件系统，数据库…… 几乎都是很蹩脚的设计。所谓的“Unix 哲学”，也就是进程间通信主要依靠无结构字符串，造成了一大批过度复杂，毛病众多的工具和语言的产生：AWK，sed，Perl，…… Unix 的内存管理是按“页”而不是按“结构”分配，相当于把内存分配的任务完全推给应用程序。而且允许任意的指针操作，这就像给每个老百姓一把爱走火的枪。可是又想要“安全”，自相矛盾。没办法，不得不强制进程数据空间完全隔离，使得进程间无法直接传递数据结构。进程和线程上下文切换开销过大，造成了使用大规模并发或者分布式计算的瓶颈，导致了 goroutine 和 node.js 等“变通方法”的产生。把数据无结构的存储在文件里，无法有效的查找数据，造成了关系式数据库等过度复杂的数据解决方案的产生。再加上后来 WEB 的设计，现在的网站基本上就是补丁加补丁，一堆堆的 hack。

“Unix 哲学”貌似也有好的部分，比如“每个程序只做一件事，多个程序互相合作。”然而，这个所谓的哲学其实就是程序语言（比如 Lisp）里面的模块化设计。它当然是好东西，然而这些思想被 Unix 偷来之后，有其名而无其实。很少有 Unix 程序真正只做一件事的，而且由于字符串这种通信机制的不可靠，它们之间其实不能有效地合作。有时候你换了一个版本的 make 或者 sed 之类的工具，你的 build 就莫名其妙的出问题。这就是为什么有的公司请了专门的所谓“build engineer”，因为高级别的程序员不想为这些事情操心。Lisp 程序员早就明白这个道理，所以他们尽一切可能避免使用字符串。他们设计了 S 表达式，用于结构化的传输数据。实际上 S 表达式不是“设计”出来的，它是每个人都应该首先想到的，最简单的可以表示树结构的编码方法。Lisp 的设计原则里面有一条就是：Do not encode。它的意思是，尽量不要把有用的数据编码放进字符串。Unix 的世界折腾来折腾去，XML，CORBA，…… 最后才搞出个 JSON，然而其实 JSON 完全不如 S 表达式简单和强大。Unix 就像一个脑瘤，它让人们放着最好的解决方案几十年不用，不断地设计乌七八糟的东西用来取代乌七八糟的东西。这些垃圾对人有很大的洗脑作用。前段时间我说 S 表达式比 JSON 简单，有人居然跟我说 JSON 好些，因为它结构的 field 是“无顺序”的。这让我相当无语，因为一个编码方式有没有顺序完全取决于你如何解释它。从这个意义来讲，S 表达式可以是有顺序，也可以是没有顺序的。

Unix 喜欢打着“自由”和“开源”的旗号，可是它的历史却充满了政治，宗教，利益冲突和对“历史教科书”的串改。几乎所有操作系统课本的前言都会提到 Unix 的前身 Multics，而提到 Multics 的目的，都是为了衬托 Unix 的“简单”和伟大，接下去基本上就是按部就班的讲 Unix 的设计，仿佛 Unix 就是世界上唯一的操作系统一样。 课本会告诉你，Multics 由于设计太复杂，试图包罗万象，最后败在了 Unix 手下。可是如果你仔细了解一下 Multics 的历史，就会发现最后一台 Multics 机器直到 2000 年还在运行，拥有 Unix/Linux 到现在还没有的先进而友好的特性，并且被它的用户所爱戴。Multics 的设计并不是没有问题（对比一下 Lisp Machine 和 Oberon），但是相比之下，Unix 的设计一点都不简单。Unix 抄了 Multics 最好的一些思想，有些没有抄得像，然后又引入了很多自以为聪明的糟粕。可是 Unix 靠着自己病毒一样的特征，迅速占领了市场。Unix 最开头是开源和免费的，但是后来 AT&T 发现这里面有利可图，所以就收回了使用权，并且开始跟很多人打官司。AT&T 的邪恶比起微软来，真是有过之而无不及。

Unix 的很多设计是如此龌龊，很多人却又由于官僚的原因不得不用它。以至于 Unix 出现的早期怨声载道，有人甚至组织了一个 mailing list 叫“Unix 痛恨者”(Unix Haters)。你很有可能把这些人当成菜鸟，可是这些人其实都用过更好的操作系统，有的甚至设计实现过更好的操作系统甚至程序语言。最后他们的叫骂声被整理为一本书，叫做 Unix Hater’s Handbook。让人惊讶的是，这本书有一个“反序言” (anti-foreward)，作者正是 Unix 和 C 语言的设计者之一，Dennis Ritchie。这个反序言说，Unix 这座设计缺乏一致性的监狱会继续囚禁你们，聪明的囚犯会从它里面找到破绽，可惜的是自由软件基金会（FSF）会建造跟它完全兼容的监狱，只不过功能多一些。拥有三个 MIT 学位的记者，微软的研究员，Apple 的高级科学家可能还会对这座监狱的“规矩”贡献一些文字。从这些文字里，我看到了一个炫耀武力的暴君，看到了赤裸裸的权威主义和教条主义。

可惜的是在软件的世界里任何糟糕的设计都可以流行，只要你的广告做得好，只要你的传教士够多。一知半解的人（比如十年前的我）最喜欢到处寻找“新奇”的东西，然后开始吹嘘它们的种种好处，进而成为它们的布道者。再加上大学计算机系的“紧跟市场”的传统，不幸的事情发生了：Unix 和它的后裔们几乎垄断了服务器操作系统的市场。由于 Unix 的垄断，现在的软件世界基本上建立在一堆堆的变通之上，并且固化之后成为了“珍珠”。公司里，学校里，充满了因为知道一些 Unix 的“巧妙用法”而引以为豪的人，殊不知他们知道的只是回避一些蹩脚设计的小计俩。程序员有太多的特例和细节需要记忆，不但不抱怨，还引以为豪。很少有人想过如何从根本上解决问题，历史的教训很少有人吸取，以至于几十年前犯过的设计错误还在重现。Unix 的最大贡献，恐怕就是制造了大量的工作岗位—因为问题太多太麻烦，所以需要大量的人力来维护它的运行。

现在看来，Unix 当初就是依靠《皇帝的新装》里织布工的办法封住了大家的嘴。皇帝的织布工们说：“愚蠢或者不称职的人都看不见这件衣服。”Dennis Ritchie 说：“Unix 是简单的，但只有天才才能理解这种简单。”看出来了吗？你不敢说 Unix 的设计太乱太复杂，因为这话一出口，立马会有人引用 Dennis 的话说，是你自己不够天才，所以不理解。当然了，这就意味着他比你聪明，因为只有天才才能理解这种简单嘛。哎，这种喜欢显示自己会用某种难用工具的人实在太多了。你不敢批评这些工具对用户不友好，因为你立即会被鄙视为菜鸟。

Dennis Ritchie 去世了。死者长已矣，可是有些他的崇拜者在那个时候还要煽风点火，拿他的死与 Steve Jobs 的死来做对比，把像这样的照片四处转帖，好像 Steve 死错了时间，抢了 Dennis 的风头似的。然后就有人写一些这样的文章，把世界上的所有系统，所有语言都归功到 Dennis 和 Unix 身上。看到这些我明白了，所谓的“天才”就是这样被造出来的。在我看来这些是很滑稽的谬论，就像是在说有人拿一把很钝的剪刀做出了一件精美的衣服，所以这剪刀立下了汗马功劳。其实这人一边裁布一边在骂这剪刀，心想妈的这么难用，快点做出这衣服，卖了钱买把好点的！

用了这么久 Apple 的产品，平心而论，虽然它们并不完美，然而它们并不是 Unix 的翻版，它们做出了摆脱 Unix 思想束缚的努力。它们本着机器为人服务的原则，而不是把人作为机器的奴隶。Mac 的很多内部设计跟 Unix 有着本质的不同。然而就是这样的系统，被 Dennis Ritchie 在他的反序言里面蔑称为“以 Sonic the Hedgehog 作为智力主题和交互设计基础的系统”。

有谁知道，在那同样一段时间里，Lisp 的发明者 John McCarthy，ML 的发明者 Robin Milner，都相继去世了呢？那个时候我只是在 mailing list 看到有人发来简短的消息，然后默默地思念他们给我带来的启迪。我们没有觉得 Steve Jobs 的死抢了他们的风头，因为他们不需要风头。死就是要安安静静的，让知己者默哀已经足矣。出现这种事情恐怕不能怪 Dennis Ritchie 自己，然而这些 Unix 的崇拜者们，真的应该反省一下自己的做法了。

Unix 的设计者们曾经在我的心里占据了一席之地，可是现在觉得他们其实代表了反动的力量，他们利用自己的影响力让这些糟糕的设计继续流传，利用人们的虚荣心，封住大部分人的嘴，形成教条主义，让你认为 Unix 的设计是必须学习的东西。很多人成为了 Unix 的传教士和跟屁虫，没有什么真实水平，就会跟着瞎起哄，把 Unix 设计者的话当成教条写进书里。可是他们的权威和名气是如此之大，让我在很多人面前只能无语。

Go 语言

现在，同样这帮 Unix “大牛”们设计了 Go 语言，并且依仗自己的权威和 Google 的名气大力推广。同样的这帮跟屁虫开始使用它，吹捧它，那气势就像以为 Go 可以一统天下的样子。真正的程序语言专家们都知道，Go 的设计者其实连语言设计的门都没摸到。这不是专家们高傲，他们绝不会鄙视和嘲笑一个孩子经过自己的努力做出一个丑陋的小板凳。他们鄙视，他们嘲笑，因为做出这丑陋小板凳的不是一个天真的小孩，而是一些目空一切的人，依仗着一个目空一切的公司。他们高举着广告牌，试图让全人类都坐这样丑陋的板凳。

跟当年设计 Unix 时一个德行，不虚心向其它语言和系统学习经验教训，就知道瞎猜瞎撞。自己想个什么就是什么，但其实根本就不知道自己在干什么。把很多语言都有的无关紧要的功能（比如自动格式化代码）都吹嘘成是重大的发明，真正重要的东西却被忽略。Go 语言的设计在很多方面都是历史的倒退，甚至犯下几乎所有其他语言都没有的低级错误。在语法上大做花样，却又搞得异常丑陋，连 C 和 Java 都不如。自己不理解或者实现难度大点的东西就说是不需要的，所以连很多语言支持的 parametric type（类似 Java generics）都没有，以至于没法让程序员自定义通用数据结构，只好搞出一堆特例（比如 map，make，range）来让程序员去记。这些做法都跟 Unix 如出一辙。

Go 语言最鲜明的特征就是 goroutine，然而这个东西其实每个程序语言专家都知道是什么。有些语言比如 Scheme 和 ML 提供了 first-class continuation（call/cc），可以让你很容易实现像 goroutine 这样的东西，甚至实现硬件中断的“超轻量线程”。至于 Go 那种“基于接口”的类型系统设计，我在很多年前就已经试验过，并且寄予了很大的希望。结果最后经过很多的研究和思索后发现有问题，于是放弃了这个想法。很显然，我不是第一个在这个问题上失败的人，很多语言专家在使用 parametric type 以前都试图过做这种基于接口的设计，结果最后发现不是什么好东西，放弃了。然而 Go 的设计者却没有学到这些失败教训，反而把它当成宝贝。一个很显然的问题是，在 Go 里面你经常会需要使用“空接口”（interface{}），用来表示所有类型。这就像使用 C 的 void 指针一样，有着静态类型系统的麻烦，却失去了静态类型系统的好处。

每当你提到 Go 没有 parametric type，Go 的拥护者们就说“我看不到这有什么用处”，就像一些非洲土著跟你说“我看不到鞋子有什么用处”一样。他们利用人们对 Java 的繁复和设计模式的仇恨，让你抛弃了它里面的少数好东西。其实 Java generics 不是 Java 首先有的。它的主要设计者其实包括 Haskell 的设计者之一 Philip Wadler。这种 parametric type 很早就出现在 ML，Haskell 等语言里面，是非常有用的东西。

每当受到批评，Go 的拥护者们就托词说，Go 是“系统语言”（systems language）。这里潜在的前提就是，认为 Unix 就是唯一的“系统”，而 C 就是在 Go 以前唯一的“系统语言”，好像其他语言就写不出所谓的“系统”似的。而事实是，在 C 诞生十年以前，人们就已经在用 Algol 60 这样的高级语言来写操作系统了。由于先天不足却又大力推广，所以 Go 的很多缺陷基本已经没法修补了。这样的语言一旦流行起来就会像 Unix 一样，成为一个无休止的补丁堆。如果像 Java 或者 Haskell 这样的语言还值得批评的话，对 Go 语言的设计者我只能说，去补补课吧。

Cornell

可是权威和名气的威力还是很大的。虽然 Knuth 在我心目中的位置不再处于“垄断地位”，世界上可以占据我心里那个位置的人和事物还很多。在离开清华之后我申请了美国的大学。也许是天意也许是巧合，只有两所大学给了我 offer：Cornell 和 Indiana，而我竟然先后到了这两所大学就读。

说实话，Indiana 给了我比 Cornell 更好的 offer。Cornell 给我的是一个 TA 的半工读职位，而 Indiana 给我的是一个不需要工作白拿钱的 fellowship。说实话我从来没有搞明白 Cornell 这样的“牛校”怎么会给我这样的人 offer，GPA 一般，paper 很菜，而 Indiana 却是真正在乎我的。Indiana 的 fellowship 来自 GEB 的作者 Doug Hofstadter。他从 email 了解到我的处境和我渴求真知的愿望之后，毅然决定给我，一个素不相识的人写推荐信。后来我才发现那 fellowship 的资金也是他提供的。

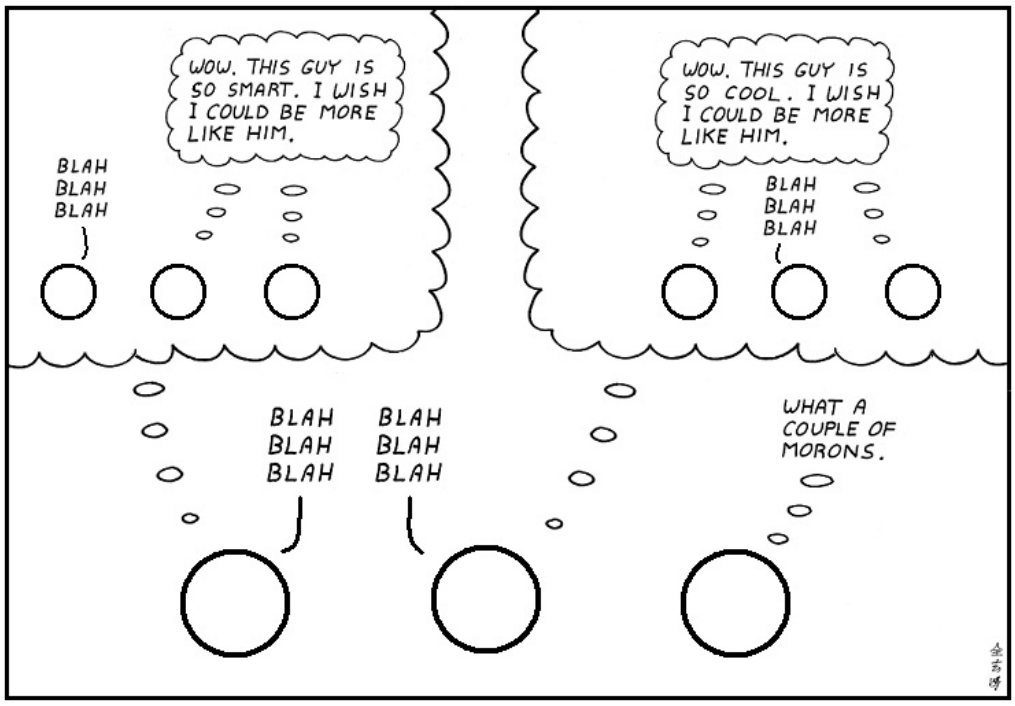
可是 Indiana 和 Hofstadter 的名气哪里能跟 Cornell 的号称 “CS前五” 相比啊？Indiana 的 offer 晚来了几天。当收到 Indiana 的 offer 时，我已经接受了 Cornell。Hofstadter 很惊讶也很失望，因为他以为我一定会做他的学生，可是听说我接受了 Cornell 的 offer，他也不知道该怎么办。我只隐约的记得他告诉我，学校的排名并不是最重要的东西……

名气和权威的力量是如此之大，它让我不去选择真正欣赏我并且能给我真知的人。有时候回想起来，我当时真的是在寻找真知吗？我明白什么叫做真知吗？

Cornell 给了我什么呢？到现在想起来，它给我的东西恐怕只有教训，很多的教训。TA 的工作可不是那么好做的，基本就是苦力，你甚至会怀疑他们录取你就是为了利用你的廉价劳动力。我第一次做 TA 就是一个 200 多人在阶梯教室上的大课，教最基本的 Java 编程。虽然有好几个 TA，但任务还是很繁重。讲课的人不是教授，而是专职的讲师。这种讲师一般得靠本科生的好评来谋生，所以虽然在学术上没什么真本事，对学生真可谓是点头哈腰，服务周到。这就苦了各位 TA 了，作业要你设计，还要设计得巧妙，要准备好标准答案，之后还要批作业，批得你头脑麻木，考试要监考，之后还要批试卷。每周还得抽好几个小时来做 office hour，给学生答疑。然后你还有自己要上的课，自己的作业，自己的考试。每当考试的时候都很紧张，因为你得准备自己的考试，还要为学生的考试多做很多工作。

如果真的学到了东西，这么辛苦也许还值得，可是那些教授真的是想教会你吗？有人打了个比方，说 Cornell 说要教你游泳，就把你推到水池里，任你自己扑腾。当你就要扑腾上岸时，他在你头上用榔头一砸，然后继续等你上岸。当你再次快要扑腾上岸时，他又举起一块大石头扔到你头上，这样你就可以死了，可是 Cornell 仍然等着你游上岸……这就是对我在 Cornell 的经历的非常确切的比喻。

我在一篇老的博文里面提到过，Cornell 的学生，包括博士生，一上课就抄笔记，一天到晚都在赶作业。可其实 Cornell 不只是爱抄笔记的学生的天堂，而且是崇拜权威者的天堂。即使你不是那么的崇拜权威，你不可避免的会被一群像朝圣者一样的人围在中间，在你耳边谈论某某人多么多么的牛。不管你向同学打听哪一个教授，得到的回答总是：“哇，他很牛的！” 然后你就去上了他的几节课，觉得不咋的嘛，可是人家就说那是因为你不理解他的价值。这种气氛我好像在另一个地方感觉到过呢？啊对了，那是在 Google。这样的气氛也许并不是偶然，Cornell 的大部分 PhD 同学当时的最大愿望，就是毕业后能去 Google 工作。当然，后来 Facebook 上升成为了他们的首选。值得一提的是，Indiana 其实是更有个性的地方。我在 Indiana 的同学们一般都把去 Google 工作作为最后的选择之一。有一次一个刚来不久的学生问，如何才能进入 Google 工作？有个老教授说，那个容易，Google 招收任何能做出他们题目的人！



Cornell 的研究可以用“与时俱进”来形容，什么热门搞什么。当时 Facebook 和社交网络正在“崛起”，所以系里最热门的一个教授就是研究社交网络的。我去听过他几堂课，他用最容易的图论算法分析一些社交网络数据，然后得出一些“理论”。其中好些结论实在太显然了，我觉得街上的卖菜大妈都能猜到，还不如研究星际争霸来得有意思点。可是 Facebook 名气之大，跟着这位教授必然有出路啦，再加上有人在耳边煽风点火，所以有好多的学生为做他的 PhD 挤破了头皮，被刷下来的就只好另投门路了。每次新来一个教授都会被吹捧上天，说是多么多么的聪明，甚至称为天才。然后就有一群的人去上他的课，试图做他的学生。结果人家每节课都是背对学生面朝黑板，喃喃自语，写下一堆堆的公式和证明，一堂课总共就没回过几次头。下面的人当然是狂抄笔记，有的人甚至带着录音笔，生怕漏掉一句话。上这样的课还不如干脆把板书打印出来让大家自己回家看。人多了竞争也就难免了。上课的同学们就开始勾心斗角，三国演义的战术都拿出来了。作业做不出来就来找你讨论，等你想讨论了就说自己也没做出来。没听懂偏要故作点头状，显得听懂了，让你觉得有压力。自己越是喜欢的教授就越是说他不咋的，扯淡，然后就自己去跟他。自己不喜欢的教授就告诉你他真是厉害啊，只可惜人家不要我。直到两年后我离开 Cornell 之前，还有好些同学因为没找到

教授而焦头烂额。因为两年内没有找到导师的 PhD 学生，基本上等于必须退学。

当我离开 Cornell 之后，有一位国内的学生给我发 email 套磁（从系里主页上找到我的地址），问我 Cornell 情况如何。我告诉他我都已经走人了，并且告诉了他我的感觉，一天到晚抄笔记赶作业之类的。然后又问我一个刚毕业的 PhD 的情况，我说他水平不咋的，博士论文我看过了，很扯淡，解决一个根本不存在的问题。他对我说的话有点惊讶，但还是将信将疑。为了确保万无一失，他在 visiting day 的时候专程去 Cornell 考察了一下。回去又给我 email，说见到好多牛人啊，大开眼界，哪里像你说的那么不堪。还说跟那位 PhD 的导师谈过话，真是世界级的牛人那，他的博士论文也是世界一流的。我就无话可说了，仁者见仁，智者见智，随他去吧，哎。

结果两年之后，我又收到这位同学的 email，说他在 Cornell 还没找到导师，走投无路了，问我有没有办法转学。

图灵奖

说到这里应该有人会问这个问题，我是不是也属于那种没找到导师走投无路的人。答案是，对的，我确实没有在 Cornell 找到可以做我导师的人。然后我就猜到有人会说，就知道王垠水平不行嘛，没搞定导师，被迫退学，哈哈！可是事情其实没他们想象的那么简单。作为一个 PhD 学生，不仅必须精通学术，而且要懂得政治和行情。哦错了，其实不精通学术也行的，但是一定要懂得政治和行情！可是由于学生之间的窝里斗，他们之间的信息互通程度，是没法和教授之间的信息互通程度相比的。这就造成了“学生阶级”在这场信息战上的劣势，总是被动的被教授挑选，而不能有效地挑选适合自己的教授。

进入 Cornell 之后我上了一门程序语言的课，就开始对这些东西入迷。可是由于“与时俱进”，Cornell 的研究方向并不是那么平衡的发展的，其实是很畸形的发展。程序语言领域的专家们早已因为受到忽视而转移阵地，剩下一群用纸和笔做扯淡理论的。说实话，在历史上程序语言方向曾经是 Cornell 的强项，出现了一些很厉害的成果。可是当我在 Cornell 的时候，只剩下两个名不见经传的教员，一个助理教授，一个副教授。其实 Robert Constable 也在那里，可惜的是他做了 dean 之后已经没空理学生了，以至于我两年之后都不知道这个人的存在。我当时也不知道 Cornell 有过这段历史，看不到它的研究重心的移动趋势。

我不喜欢那个副教授搞的项目，大部分是在 Java 上面加上一些函数式语言早就有的功能。可是人家做的是热门语言，所以拉得到资金，备受系里亲睐，他的学生们也比较趾高气昂。初次见面的时候，我跟他的一个学生说了我的一个想法，他说：“你那也能叫研究吗？待会儿我给你看看什么是真正的研究！” 其实那只是我的一个微不足道的想法，我也没说那是研究啊。只是随便聊一下而已就这么激动 -_- 何况你们那些 Java 的东西能算是研究？我是不可能跟那样的人合作的，所以我就跟那个助理教授做了一点静态分析的项目。当然我们分析的也不是什么好东西，是用 Fortran 写的 MPI 程序。不过说实话，那个助理教授其实挺有点真知灼见，他有几句话现在仍然在指引我，防止我误入歧途。其中一句话是针对我对 π-calculus 的盲目崇拜 说的：“那些理论其实不管用的。最好是针对自己的问题，自己动脑筋想。” 他也是很谦虚很善良的人，可是好人不一定有好报的。后来他没有拿到 tenure 职位，不得不离开 Cornell 加入了工业界，而我就失去了最后一个有可能在程序语言方向做我的导师的人。

没办法，我就开始探索其它相关领域的教授，比如做数据库的，做系统的，看他们对相关的语言设计是否感兴趣。可惜他们都不感兴趣，而且告诉我程序语言领域太狭窄了。我当时还将信将疑，甚至附和他们的说法，可是现在我断定他们都是一知半解胡说八道。如果这些人虚心向程序语言专家请教，现在数据库和操作系统的设计也不会那么垃圾，关系式，SQL，NoSQL，…… 一个比一个扯淡。没有办法，我就开始探索其他的方向，开始了解图形学和数值分析等东西，进展很不错。可是终究我还是发现，我不喜欢图形学和数值分析所用的语言。我想制造出更好的程序语言来解决这些问题。可是跟教授们谈这些想法的时候就感觉是在对牛弹琴，他们完全不能理解。后来我发现，教授们貌似不喜欢有自己想法的学生，他们更希望找到愿意“打下手”的学生，帮助实现他们自己的想法。

这就让我走到了跟那位向我打听 Cornell 情况的同学差不多的局面，真是心里有许多的苦却没有人可以理解。这时候我想到了系里的一些德高望重的教授，比如得过图灵奖的人，也许这些顶级的大牛会给我指出方向。于是我就联系到一位图灵奖得主，说想找他聊聊。我说我感兴趣的东西 Cornell 貌似并不重视和发展。Cornell 的校训是“any person, any study”，而我想 study 的东西却得不到支持。最后我谈了一下我对 Cornell 的总体感受。我说我觉得大家上课死记硬背，不是很 intellectual，我不是很确定学术界是否还保留有它原来的对智慧和真知的向往。

我很诚恳的告诉了他这些，只是希望得到一些建议。结果他不但没有理解任何一点，而且立马开始用质问的语气问我，你成绩怎么样？考试都通过了没有？哎，说白了就是想搞清楚你是不是成绩不好没人要。怎么就跟高中教导主任一样。于是乎那次谈话就这样不了了之。可是没有想到，这次谈话就造成了我最后的离别。在学生们互相之间勾心斗角，不通信息的同时，系里的教授们其实背后都是“通气”的。他们根本不懂得如何教学，就知道拿作业和考试往学生头上砸，幸存下来的就各自挑去做徒弟，挨不住的就打发掉。这算盘打得真是妙啊。我也不知道他们是什么机制，每个学生对哪些教授感兴趣，表现如何，他们貌似都了如指掌，貌似背后有个什么情报网。然后系里的教授们不知道怎么的，仿佛就都知道有这样一个不知趣的学生，居然敢说学术界的坏话！

大地震前夕的天空总是异常的美。我竟然在过道里看到那位图灵奖教授对我点头致意并且微笑，以前做 TA 时把我呼来唤去还横竖不满意的教授也对我笑脸相迎。我仿佛觉得那一席话打动了那位德高望重的教授，再加上在图形学和数值计算的扎实进展，也许我的学术生涯有了转机。可是，我那一次真正的领悟了什么叫做所谓的“笑里藏刀”。

由于那个学期上的图形学还有矩阵计算的课成绩都不错，我心想应该能找这两门课的授课教授的其中一个做导师吧。再加上那些貌似友好的笑容…… 所以没想很多，居然过了一个非常快乐的寒假。没有任何前兆，没有任何直接的通知（email，电话），一封纸信不知道是什么时候默默地进到了我在系里的“信箱”—一个我基本上从来不看的，系里用来塞广告信息的信夹子里，直到下一个学期开始的时候（2月份）我才发现。信是系主任写的，大概就是说，由于你的表现，我们觉得 Cornell 不是适合你的地方……

说得对，我也觉得 Cornell 不适合我。我本来就有想走的意思，可我一般呆在一个地方就懒得动。如果你们早一点告诉我这个，比如12月以前，我还可以申请转学到其它学校。可是都 2 月份了才收到这样的东西，Cornell 啊 Cornell，你让我现在怎么办？我想我可以说你不仁不义吧？

在这个万分窘迫的时候，我想起了曾经关心过我却又很失望的 Hofstadter。我告诉他我在 Cornell 很不开心，我很想研究程序语言，可是 Cornell 不理解也不在乎这个领域。他回信说，没有关系，你能找到自己喜欢的东西就应该去追寻它。Indiana 的 Dan Friedman 正好是做程序语言的，你可以联系他，就说是我介绍你去的。

于是给 Friedman 发了 email，很快得到了回信说：“Yin，两年前我们都看过你的材料，我们觉得你是非常出众的学生，可惜你最后没有选择我们。你要明白，人生最重要的事情不是名利，而是找到你愿意合作的人。你的材料都还在我们这里。现在招生已经快结束了，但是我会把你的材料提交给招生委员会，让他们破例再次考虑你的申请。” 我和 Dan Friedman 的故事就从这里开始了。

我在 Cornell 的遭遇貌似不可告人的耻辱和秘密，然而我今天却可以把它公之于世，因为 Cornell 不再有任何资格来评价我。依靠自己的努力和 Indiana 的老师们的培养，我的水平已经超越了 Cornell 计算机系的大部分教授。现在我觉得自己就像那个到 Cornell 学“游泳精髓”人，本来就是会游泳的，可是每到岸边 Cornell 就搬起大石头来砸我，还说我不会游。于是我钻到水底下钻了一个洞，把水放干。

由于曾经与多位图灵奖得主发生不大愉快的遭遇，再加上在自己的研究中多次受到其它图灵奖得主的理论的误导，而且许多位图灵奖得主最主要的贡献仍然在给软件行业带来混乱，图灵奖这个被许多计算机学生膜拜的神物，其实在我心里已经没有任何效力了。很多人可能对此难以想象，可是对图灵奖是这种态度的不只我一个人。我认识的几乎所有程序语言专家几乎都不拿图灵奖当回事，而且其中很多人甚至不拿图灵本人当回事，觉得他设计了一些非常丑陋的东西。虽然我现在觉得图灵的研究成果确实有一定价值，但由于上面的原因，拿图灵奖来开玩笑还是成为了我的家常便饭。我甚至觉得 ACM 应该停发这个奖，因为它是一种非常虚幻和政治的东西。每当人们谈起这些“大奖”煞有介事的时候，就让我看到了他们的愚昧。

常青藤联盟和“世界一流大学”

我在 Cornell 的经历应该不是偶然，不是因为我比较特殊。跟我同时进入 Cornell 的博士生有好几个没有拿学位就离开了。其中有一个是非常聪明的少年班，18岁就读 PhD 了，我根本听不懂的理论课他还能拿A。可是四年后他退学去了 Facebook，说真是太难毕业了，神马都是扯淡。有些本科生也告诉我类似的经历，说被一个叫做“笑面虎”的教授“整了”。Cornell 的自杀率居美国大学前列。离开以后的有一天，忽然看到新闻报道说一周之内有三个 Cornell 学生从瀑布旁边的那座桥跳下去，结果派了警察在桥上日夜巡逻。我觉得自己在 Cornell 所感受到的压力确实超乎想象，是有可能把人逼上绝路的。现在回想起来真是可笑，因为下意识里在乎权威和名气，我给予了一群根本没有资格来教育我的人莫大的权力，让他们可以向我施加无端的压力。

应该指出，这种现象应该不是 Cornell 所特有的。我对清华，还有 Princeton，Harvard，MIT，Stanford，Berkeley，CMU 等学校的学生都有了解。这些所谓的“世界一流大学”或者“世界一流大学 wannabee”差不多都是类似的气氛。你冲着它们的名气和“关系网”挤破了头皮进去，然后就每天有人在你耳边对其它人感叹：哇，他好牛啊！发了好多 paper，还得了XX奖。跟参加传销大会似的，让你怀疑这些人还有没有自尊。然后就是填鸭式的教育，无止境的作业和考试，让你感觉他们不是在“教育”你，而是在“筛选”你。这种筛选总是筛掉最差的，但也筛掉最好的。因为最好的学生能意识到你在干什么，他们不给你筛选他们的机会。一旦发现其实没学到东西，中途就辍学出去创业了。所以剩下来的就是最一般的，循规蹈矩听话的。在这样的环境里，你感觉不到真正的智慧和真知的存在。GRE 考试所鼓吹的什么“批判性思维”（critical thinking ）在美国大学里其实是相当缺乏的。学生们只不过是在被培训成为某些其他人的工具，他们具有固定的思维定势，像是一个模子倒出来的。他们不是真正的创造者和开拓者。

人们在这些大学里的时候都是差不多感受的，可是一旦他们出来了，就会对此绝口不提。自己身上挂着这些学校的镀金牌子，怎么能砸了自己的品牌，长别人的威风？所以每当我批判 Cornell 就有些以前的同学一脸的着急相，好像自己没有吃过那苦头一样。

程序语言专家

虽然我在 Indiana 得到了思想的自由，但这种自由其实是以孤独为代价的。我并不是一个自高自大不合群的人，但是我不喜欢跟一群像追星族一样的人在一起。应该说在 Indiana 的日子里，权威主义的影子也是经常出现的。Indiana 学生们的权威比较特殊一点，不然就是 Dan Friedman，不然就是 Kent Dybvig。Friedman 的身边总是围绕着一群自认为是天才的本科生，喜欢拍他的马屁，喜欢在人面前炫耀。博士生们开始时貌似还比较酷，可是后来发现其实也有很多类似现象，急于表现自己，越是研究能力弱的人越是爱表现。所以你就发现有人开头为了混进这个圈子拍你的马屁，过了两年就开始自高自大，而且经常想这样来压倒你：“Kent 说过……”我很尊敬 Dan 和 Kent，但我其实在很多方面已经超越了他们。我看到他们的一些思维方式并不是那么的正确，我也从来不引用他们的话作为理论依据。对权威的崇拜其实显示了一个人心理的弱小。如果你对自己有信心，有自己的想法和判断力，又何必抬出个名人来压制别人呢？

在我自己心里毫无疑问的是，我是 Indiana 最厉害的程序语言（PL）学生。由于我不断地动手尝试新的想法，所以几乎没有任何其他人的研究逃脱过我的探索。我从来不记录自己的半成品和失败（因为太多了），而且我对自己的标准异常的高，所以我经常看到有人做演讲或者写论文，里面其实是我很久以前尝试过又抛弃了的想法。有时候我去听别人的演讲，就会立即看出破绽，问一些演讲者答不出来的问题。其实很多时候我只是怀疑自己，我试图给那些想法再一次的机会来证明它们的价值，而且问得相当委婉，但那样的问题仍然是不受欢迎的，所以同学们甚至一些助理教授

看到我在场都是心惊胆战的。吃饭的时候我也不喜欢旁边的人讨论问题，因为他们经常显示出对理论提出者的膜拜心理，而且煞有介事，可惜那些经常是我早就知道不管用的理论。他们有时候其实也知道那些是扯淡的，但却又怕我捅破这窗户纸，所以就像鸵鸟一样把头埋在沙子下面。

我也想合群一点，但是屡试不爽，所以后来我就基本是孤立的做自己的研究了。最开头是不得已，但后来就越来越喜欢独自一人。这是不可避免的，因为创造力和孤独几乎是双胞胎。因为免去了跟人讨论的时间，我有了大把的时间来做自己的探索。然后我才发现当年期望的那种 common room 其实没什么用，因为那里根本不会有人理解你在说什么。现在即使有这样的地方我也不会去了。

我从一开始进入 Indiana 就没想过要拿博士学位，我只是在玩弄这个系统以达到我求知的目的。所以除非危及到我的存在，我把学校对学位的各种要求都抛到了九霄云外。给教授做 RA 几乎总是被要求研究各种毫无前途的东西，与我自己的思考相冲突，所以我后来干脆都做 TA 了。虽然累点，但不怎么费脑力。其结果是，在短短的一两年时间之内，我利用自己抠出来的时间，独自摸索出了这个领域大部分的理论。我经常不看书不看论文，在一个星期之内解决别人十多年才完成的研究。让人惊讶的应该不是我有多么聪明，而是这些研究者们十年来到底在干什么。我从来不认为自己比别人聪明，我只是觉得很多人的脑子被禁锢了而已。我有非常简单的头脑，我看不懂复杂的公式，听不懂高深的术语。可正是因为这一点，让我脱离了已有理论的困扰。

可以说，这个领域在过去一个多世纪的研究，很少有逃脱过我的洞察力和直觉的。这些研究最早可以追溯到 1870 年代。我一般很少看论文，因为自己想清楚一个问题其实花不了那么多时间的。看别人的论文一般都枯燥乏味，所以与其花那么多时间读论文还不如自己思考。当我看论文的时候，一般是想搞清楚自己琢磨出来的问题有没有人已经研究过了，所以很多论文只需要扫一下就够了。我看到一个东西一般很快就会知道它到底会不会管用。我经常发现一些被认为很艰深的理论其实是在解决根本不存在的问题，甚至是在制造问题，而真正的问题却没有得到有效的解决。很多问题其实是权威的阴影造成的，它让人们不敢否认这些大牛思想的价值，不敢揭穿它们，抛弃它们，甚至想让自己寄生在它们上面，所以很多的时间花在了解决一些历史遗留问题，而不是真正的问题。这就是为什么我的英文 blog 标题叫做“Surely I Am Joking”，因为它记录了一些我认为根本不存在，或者是人为造成的问题。

逻辑学家

批评 PL 领域的问题并不意味着其它领域就好一些。恰恰相反，我认为做系统和数据库的领域有更大的权威崇拜和扯淡的成分。有时候程序语言专家看起来很明显的问题，做数据库和操作系统的人却看不到，扯来扯去扯不清楚，还自以为是的认为 PL 的东西他们都懂。

程序语言的理论是计算机科学的精髓所在，可是程序语言专家有他们自己的问题：他们膜拜逻辑学家。几乎每一篇 PL 领域的论文，至少有一页纸里面排列着天罡北斗阵一样的稀奇古怪的逻辑符号，而它们表示的其实不过是一些可以用程序语言轻松做出来的解释器和数据结构。有人（比如 Kent Dybivg）早就发现了这个规律，所以写了一些工具，可以把程序语言自动转换成 LaTeX 格式的逻辑公式，用以对付论文的写作。

有人觉得那些公式有“数学的美感”，可是它们其实是挺有毛病的设计。如果你看看现代逻辑学鼻祖 Gottlob Frege 的原著，就会发现其实最早的时候逻辑学不是用公式表示的。Frege 那篇开创性的论文 Begriffsschrift 里面全都是像电路图一样的图片，只有 20 多页，而且非常容易读懂。不知道是哪一个后辈把电路图改成了一些稀奇古怪的符号。其实他的目的是用符号来表示那些电路图，结果到后来徒孙们以为那些符号就是祖传秘籍的精髓，忘记了那些符号背后的电路图，所以导致了今天的混乱局面。看完了 Frege 的论文，我再一次领悟到了之前那句话：跟真正的大师学习，而不是跟他们的徒弟。

ACM SIGPLAN 的主席 Philip Wadler 有一次写了一篇论文介绍 Curry-Howard corresponce，里面提到，好的点子逻辑学家总是比我们先想到。可是他却没有发现，其实程序语言的能力已经大大超越了数理逻辑，数理逻辑那些公式里面的 bug 其实不少。因为逻辑学家们不用机器帮助进行推理，有些问题搞了一百多年都搞不清楚是怎么回事，然后就弄出一些特殊情况和补丁来。有了一堆逻辑“定理”，却又不能确信它们是正确的，而且存在悖论一类无厘头的东西，所以又掰出一些 model theory 之类的东西来验证它们的正确性。逻辑学家们折腾了一百多年都是在折腾类似的事情，却没怀疑过老祖宗的设计。我之前提到的 Hindley-Milner 系统的问题，很大部分原因就在于它所使用的逻辑里面其实有一个根本性的误解。简言之，就是把全称量词 ∀ 随意乱放，导致输入与输出关系混乱。这也就是我为什么不喜欢 Haskell 和 OCaml 的最主要原因。

现在最热门的逻辑学家莫过于 Per Martin-Löf。他的类型理论 Martin-Löf Type Theory 被很多 PL 人奉为神圣。我一直没有搞清楚这个类型理论有什么特别，直到有一天我把 Martin-Löf 1980 年的那篇论文（其实是演讲稿）拿出来看了一遍。然后我发现他通篇本质上就是在讲一个 partial evaluator 要怎么写，而我早就自己写过 partial evaluator。其实并不是特别神奇的东西，只需要在普通解释器里面改一两行代码就行，可是有人（比如 Neil Jones）却为此写出了 400 多页的书和大量的论文。

之前提到的 Curry-Howard corresponce 也被很多人奉为神圣，它来自数学家 Haskell Curry 和逻辑学家 W.A. Howard 的一些早期发现。他们发现有些程序和定理的证明之间有对应的关系。然后就有 PL 专家开始走火入魔，说“程序就是证明，程序的类型就是定理”。可是他们却没有发现这个说法没法解释操作系统这种程序，因为它被设计为永远不停地运行，所以不能满足一个证明所具有的基本特征。而且很多程序被设计出来根本就不是要证明什么定理，它们是被设计来帮人做事情的。所以我觉得“程序就是证明”是很牵强附会的说法，你不能因为有的程序可以用来证明数学定理，就认为所有的程序都是某个定理的证明啊！把那句话反过来，说成“证明就是程序”还差不多。

但从以上的发现，我很高兴的看到了自己作为一个程序员的价值。很多人瞧不起程序员，把他们蔑称为“码农”，可是程序如果写好了，其实比起那些高深的逻辑学家和哲学家还要强，因为程序语言其实比数理逻辑还要强。有一位数学

家说得好：为了真正深入的理解一个东西，你应该把它写成程序。还有人说，编程只是一门失传的艺术的别名，这门艺术叫做“思考”。我觉得很在理。

再见了，权威们

几经颠簸的求学生涯，让我获得了异常强大的力量。我的力量不仅来自于老师们的教诲，而且在于我自己不懈的追求，因为机会只亲睐有准备的头脑。

曾经 Knuth 是我心中唯一的权威，后来我又屈服于 Cornell 和常青藤联盟的权威和名气。在一而再再而三的上当受骗之后，我终于把所有的权威们从我的脑子里轰了下去。也许有时候轰得太猛烈了一些，但总的说来是有好处的。不再是我心目中的权威并不等于我鄙视他们或者不尊敬他们。我只是获得了不用膜拜他们，不用跟一群人瞎起哄的自由。我不尊敬的人都是一些自视过高的人或者他们的跟屁虫。一般来说，权威们在我的脑子里失去的只是他们在很多其他人脑子里的那种被膜拜的地位，那种你可以用“XX人说过……”来压倒理性分析的地位。现在他们在我心目中是一群普通的，由蛋白质形成的生物，有好心肠或者坏心眼的，高傲，谦虚或者虚伪的人。我不会自讨苦吃，他们的想法如果真的好，我当然要拿来用，但是没有任何人的东西我是不加批判全盘接受的。我深深地知道接受错误想法的危害性，所以我也希望大家都具有批判的思维，不要盲目的接受我说的话。我不喜欢“大神”或者“牛人”这种称呼，我也反感那种自称膜拜我的人，因为正是这种人让权威主义现在横行于世。

美国的权威主义胜于欧洲，但也不是每个人都那么的崇拜权威，而中国才是权威主义的重灾区。像“图灵奖得主XX”这样的称呼，恐怕只有在中国才见得到。所以我希望国内的同学们，不要盲目的崇拜国外的所谓“大师”，“牛校”或者“牛公司”。祝你们早日消灭掉心里的各种权威以及对他们的畏惧心理，认识到自己的价值和力量。

后记（关于 IU）

有些人看了我的文章介绍在 IU 的经历，告诉我他们申请了 IU。我觉得有必要免责声明一下：我没想到，也不希望有人因为我的文章而去 IU，YMMV (your mileage may vary)。由于我有所准备，所以对于 Friedman 的教学如鱼得水。很多同学也说学到很多，可是有一些其他人告诉我他们觉得 Friedman 的课他们听起来很吃力，只能说是勉强过关。而且我只介绍了 IU 好的方面，却把不大好的地方一笔带过了。我在 IU 也有很艰难的时候。现在的情况是 Kent Dybvig 已经离开了 IU，加入了 Cisco。他的公司 Cadence Research Systems 和 Chez Scheme 也并入了 Cisco。Dan Friedman 由于年纪原因说不准还带不带学生。最近引进了一些貌似不错的助理教授，但是我跟他们都不熟。我的经验是助理教授一般都会为了研究资金，为了升为正教授而做一些身不由己的事情。其他的 CS 方向我都说不准 IU 是什么水平，所以还请同学们自己斟酌。我可以毫无疑问的一点是，IU 有非常美丽的校园，大大的超过清华，北大，Cornell，Stanford，MIT。