**Coccinelle: 10 Years of Automated Evolution in the Linux Kernel**

文献报告

M201877254 计算机硕1812班 罗宏宇

# 关于论文

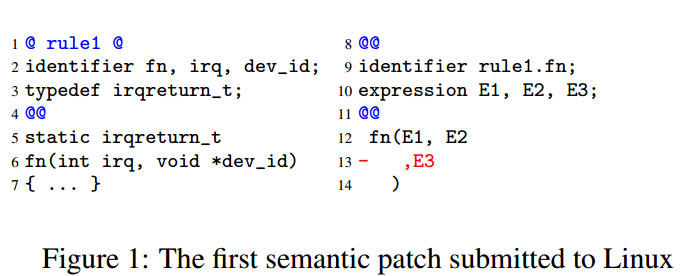
该论文是由Julia Lawall、Gilles Muller等人发布表在USENIX ATC ’18会议上。该论文主要内容是介绍了Coccinelle的设计因素以及这10年来内核开发人员和其他使用者使用Coccinelle的情况，是一篇工具科普总结性论文。

# 设计背景

最初Coccinelle的设计是为了用于解决一个特定的问题，即将Linux设备驱动程序从Linux 2.4（以前的稳定版本）移植到稳定版Linux 2.6，而这个设计的驱动力是由早期关于Linux内核中的附带演变的论文[45]推动的，即API客户端响应API接口变化所需的演进。并且针对内核开发人员，不仅必须显示地向其展示代码，并且要保证生成的代码必须保留原始源代码结构，以确保代码的持续可维护性。

# 表达性

Coccinelle提供了转换语言SmPL（语义补丁语言）和用于将SmPL语义补丁应用于C代码的引擎。一个典型例子如下所示：

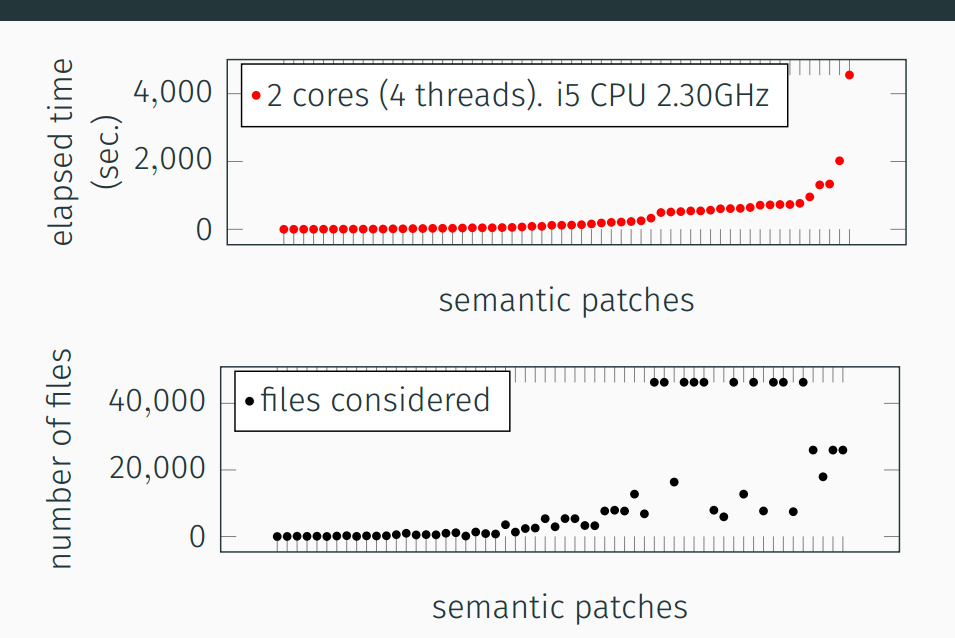


第一个名为rule1的规则声明了三个标识符元变量fn，irq和dev\_id，分别表示要匹配的函数的名称和两个参数的名称。第一个规则的其余部分是与函数定义匹配的模式，其中参数和返回值被指示为具有特定类型（第5-6行），并且允许主体是任意语句序列（第7行） 。第二个规则没有名称，声明了四个元变量：函数名称fn，其值显式继承自上一个规则（第9行），以及三个表达式元变量E1，E2和E3，表示任意参数表达式（行10）。此规则的其余部分匹配对第一个规则中标识的函数的调用。在此调用中，指示第三个参数被删除（第13行）。

**随着编程语言（如C语言）的规则变化，该工具也在不断进行改进，其中，**两个大大提高了Coccinelle表现力的演变是引入了位置变量和脚本规则。位置变量是Coccinelle元变量，它匹配文件中术语出现的位置。位置变量允许重新匹配后面的规则中的相同代码，以及确保一个规则中的匹配与先前规则中的匹配不同；脚本规则的原始动机，即发现诸如缓冲区溢出之类的错误并不成功，但现在的作用已经转为在实践中用于错误报告，计数等。

# 性能

该应用必须可在典型的开发者笔记本电脑上使用，为了减少运行时间，默认情况下，Coccinelle仅处理.c文件，只处理与.c文件位于同一目录中的头文件或与.c文件同名的头文件，并且不执行过程间分析。另一方面，Coccinelle没有宏扩展，而是使用启发式来解析宏用途。且它仅提供尽力而为型的推理，而没有其他程序分析。在这十年间，Coccinelle也进行了不少优化，一是索引，它通过使用glimpse工具以及id-utils工具体现；二是引入并行性，如parmap。另外，该论文还给出了性能的评估方法——完成Linux内核本身的编译。下图分别显示了在Thinkpad笔记本电脑上使用两个核心运行超线程处理时的语义补丁所用的时间以及使用id-utils对所考虑的文件数进行索引的影响。



# 正确性和传播策略

与其他案例不同，从转型正确性的角度来看，Coccinelle没有重大变革，它的正确性还是主要依靠开发人员对代码库的知识以及创建和改进语义补丁的简易性。而对于传播策略，则是展示工具如何有用，而不是试图强制使用它是有效的。

在提交日志中提到Coccinelle的人中，我们区分了六类Coccinelle用户：

Coccinelle开发人员。这些是Coccinelle开发团队的成员，以及团队雇用的人员来传播Coccinelle。

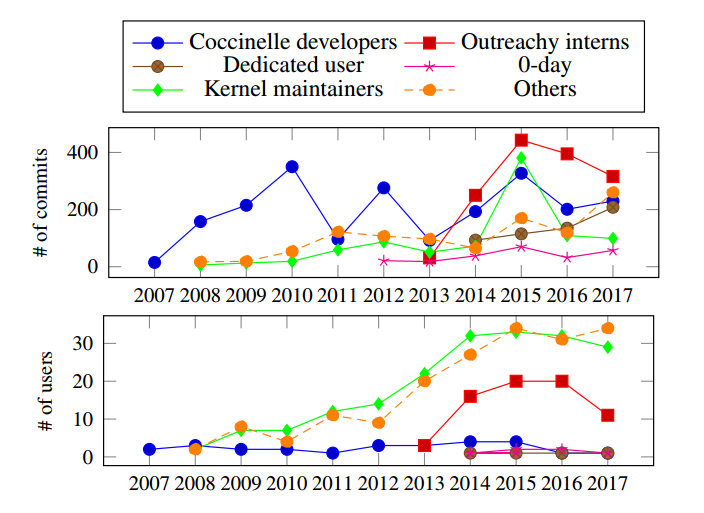
外围实习生。 Linux内核参与了Outreachy实习计划，实习生可以在申请过程或实习中使用Coccinelle。

专用用户。这是一个单独的开发人员，他在内核中使用Coccinelle进行一系列广泛相关的简单更改。

0-day测试。这是英特尔的一项自动化测试服务，可在每次提交数百个git树时为多个内核配置构建和引导Linux内核。该服务还针对每次提交的结果运行了许多静态分析工具，包括Coccinelle。

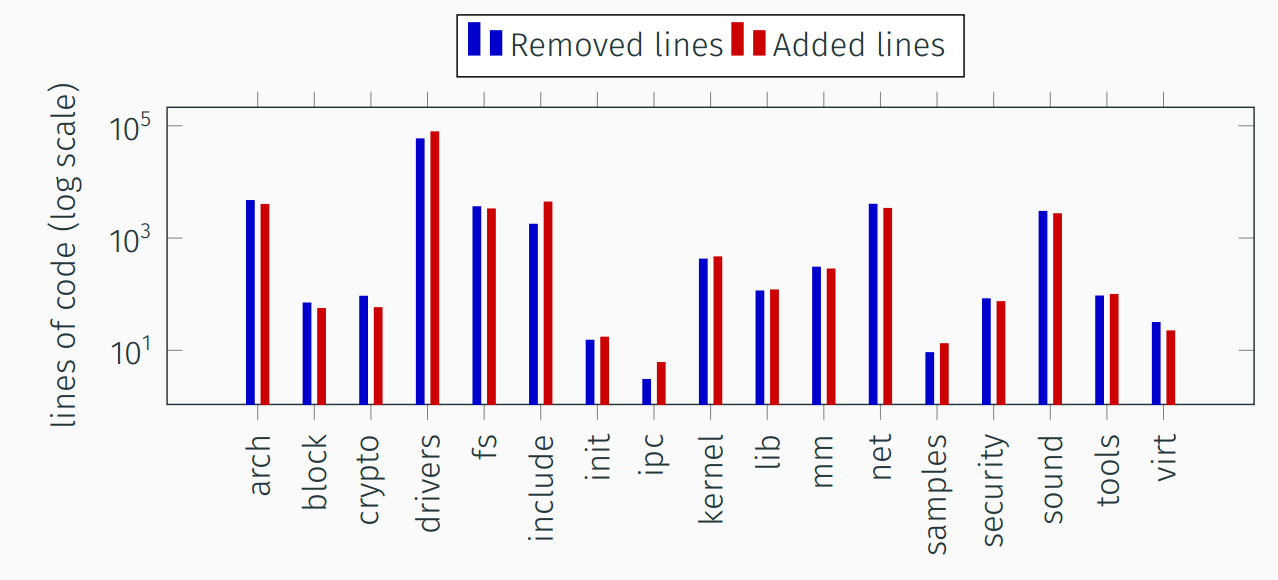
内核维护者。这些是接收和提交补丁的内核开发人员，通常负责某些子系统的持续良好状态。

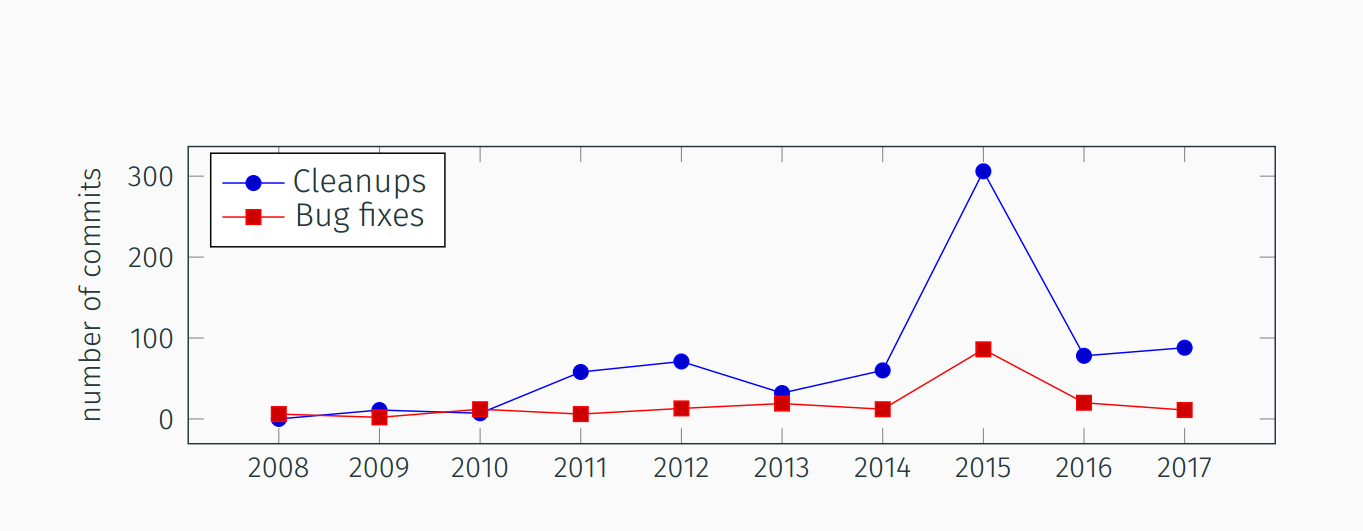
其他。这些是其他Linux内核贡献者。这些贡献者可能是频繁的或偶尔的。

下图的顶部显示了使用来自各种Coccinelle用户的Coccinelle每年提交的数量，而底部显示了每个类别中涉及的Coccinelle用户数量。

# 影响

在过去的10年中，Coccinelle已经越来越多地应用于Linux内核，Coccinelle开发人员和Linux内核开发人员都在使用它。

下图显示了在各种内核子系统中使用Coccinelle通过提交删除和添加的行数。

再来了解一下使用Coccinelle进行的更改，Coccinelle有助于在整个内核中执行更改，这可能涵盖由多个维护人员管理的代码。主要的例子有：TTY。删除未使用的函数参数；IIO。添加缺失的devinit和devexit注释；DRM。消除数据结构中的冗余字段；中断。准备从中断处理程序中删除irq参数，然后删除该参数……下图描述了当清理或错误修复时，维护者使用Coccinelle的完整补丁集数目：

另外，Coccinelle与许多Linux发行版一起打包，例如Ubuntu，Debian，Fedora，Gentoo和Archlinux，虽然Coccinelle主要用于Linux内核，但它也用于其他软件项目。 RIOT，qemu和systemd在其源代码分发中包含语义补丁。

# 总结

论文回顾了程序转换工具Coccinelle的演变及其对Linux内核的影响，Coccinelle的经验可以帮助指导其他想要对开源系统开发人员社区产生影响的项目。

首先，可见性是必要的，并考虑目标用户所具有的专业知识。

其次，该工具必须易于安装和免费提供。

第三，该工具必须易于使用且功能强大，支持快速解决用户遇到的问题。

最后，在一个研究环境中，人们总是试图让工具做得更多，直到由此产生的复杂性导致工具在其自身重量下崩溃。

# 参考文献

[1] BESSEY, A., BLOCK, K., CHELF, B., CHOU, A., FULTON, B., HALLEM, S., GROS, C.-H., KAMSKY, A., MCPEAK, S., AND ENGLER, D. R. A few billion lines of code later: using static analysis to find bugs in the real world. Commun. ACM 53, 2 (2010), 66–75.  
[2] BRUNEL, J., DOLIGEZ, D., HANSEN, R. R., LAWALL, J. L., AND MULLER, G. A foundation for flow-based program matching: using temporal logic and model checking. In POPL (Savannah, GA, USA, 2009), pp. 114–126.  
[3] ENGLER, D. R., CHELF, B., CHOU, A., AND HALLEM, S. Checking system rules using system-specific, programmerwritten compiler extensions. In OSDI (2000), pp. 1–16.  
[4] KHOROSHILOV, A., MANDRYKIN, M., MUTILIN, V.,NOVIKOV, E., PETRENKO, A., AND ZAKHAROV, I. Configurable toolset for static verification of operating systems kernelmodules.Programming and Computer Software 41, 1 (2015),49–64.  
[5] KOYUNCU, A., BISSYANDE´, T. F., KIM, D., KLEIN, J., MONPERRUS, M., AND TRAON, Y. L.Impact of tool support in patch construction. In ISSTA (2017).  
[6] LATTNER, C., AND ADVE, V. S. LLVM: A compilation framework for lifelong program analysis & transformation. In CGO(2004), pp. 75–88.  
[7] LAWALL, J., PALINSKI, D., GNIRKE, L., AND MULLER, G.Fast and precise retrieval of forward and back porting information for Linux device drivers. In USENIX Annual Technical Conference (2017), pp. 15–26.  
[8] LAWALL, J. L., BRUNEL, J., PALIX, N., HANSEN, R. R., STUART, H., AND MULLER, G.WYSIWIB: exploiting fine-grained program structure in a scriptable API-usageprotocol finding process. Softw., Pract. Exper. 1, 43 (2013), 67–92.  
[9] LOZI, J.-P., DAVID, F., THOMAS, G., LAWALL, J. L., AND MULLER, G. Fast and portable locking for multicore architectures. ACM Trans. Comput. Syst. 4, 33 (2016), 13:1–13:62.  
[10] MACHIRY, A., SPENSKY, C., CORINA, J., STEPHENS, N.,KRUEGEL, C., AND VIGNA, G. DR. CHECKER: A soundy analysis for Linux kernel drivers. In USENIX Security (Vancouver, BC, Canada, 2017).  
[11] MERILLON ´ , F., REVEILL ´ ERE ` , L., CONSEL, C., MARLET, R.,AND MULLER, G. Devil: An IDL for hardware programming.In OSDI (San Diego, CA, USA, 2000), USENIX Association.  
[12] NECULA, G. C., MCPEAK, S., RAHUL, S. P., AND WEIMER, W. CIL: Intermediate language and tools for analysis and transformation of C programs. In Compiler Construction, 11th International Conference (Grenoble, France, Apr. 2002), LNCS 2304, pp. 213–228.  
[13] PADIOLEAU, Y., LAWALL, J., HANSEN, R. R., AND MULLER, G. Semantic patches for collateral evolutions in device drivers. In Linux Symposium (Ottawa, Canada, June 2007).  
[14] PADIOLEAU, Y., LAWALL, J. L., HANSEN, R. R., AND MULLER, G. Documenting and automating collateral evolutions in Linux device drivers. In EuroSys (2008), pp. 247–260.