Shell命令错误自动修复的研究

——2020年南京大学本科毕业论文答辩

南京大学匡亚明学院
陈劭源
导师：蒋炎岩博士
2020.6.10

南京大学
NANJING UNIVERSITY

SPAR
System & Program Analysis Research

Shell命令

- Shell是操作系统提供的命令行界面
  - Shell是类Unix操作系统中常用的系统软件
  - 使用Shell命令可以方便地完成文件操作、源代码编译、系统设置配置等任务
  - Shell命令或脚本由Shell解释器执行，常用的Shell解释器有ash，bash，ksh，zsh等
  - Shell可以以多行脚本的方式执行，也可以以单行命令的方式交互式地执行

我们的scope

Shell命令中的错误

- 语法错误
  - 输入的命令不符合命令的语法规则

通常在man文档中
或-help信息中有定义

```
➜ sudo apt uninstall tmux
E: Invalid operation uninstall
```

- 语义错误
  - 符合命令的语法规则，但其含义与用户的期望不一致

```
➜ cd foo
sh: 1: cd: can’t cd to foo
```

现有工作

- Thef**k: 一个基于规则的错误修复工具
  - 拥有一个庞大的规则库，每条规则用于检测和修复一类特定错误
  - 所有规则均为手工编写
  - 例如：git_not_command规则可以检测git的子命令输入错误

```
➜ git brnch
git: 'brnch' is not a git command. See 'git --help'.

Did you mean this?
    branch

➜ fuck
git branch [enter/↑/↓/ctrl+c]
```

现有工作

- NoFAQ: 自动合成修复规则 [1]
  - 使用版本空间代数（VSA）算法 [2]
  - 仅需提供少量输入/输出样例，即可合成修复规则

```
➜ java run.java
Could not find or load main class
run.java
➜ java run
```

Example 1

```
➜ java meta.java
Could not find or load main class
meta.java
➜ java meta
```

Example 2

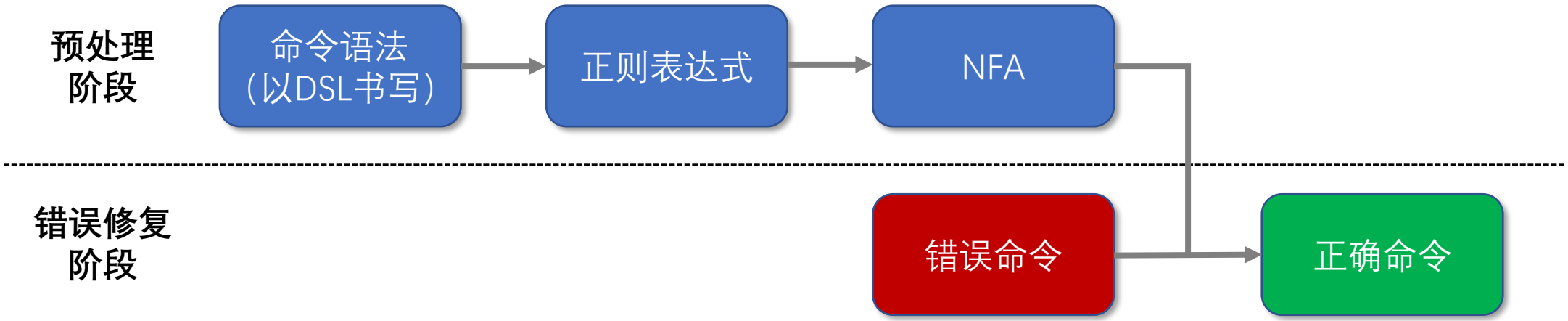
NoFAQ

$$\textbf{match}\ [\textsc{Str}(\texttt{java}), \textsc{Var-Match}(1, \epsilon, \texttt{.java})]$$
$$\textbf{and}\ [\textsc{Str}(\texttt{could}), \textsc{Str}(\texttt{not}), \textsc{Str}(\texttt{find}), \textsc{Str}(\texttt{or}), \textsc{Str}(\texttt{load}), \textsc{Str}(\texttt{main})$$
$$\textsc{Str}(\texttt{class}), \textsc{Var-Match}(2, \epsilon, \texttt{.java})] \rightarrow$$
$$[\textsc{Fstr}(\texttt{java}), \left\{ \begin{array}{c} \textsc{Sub}(\textsc{Ip}(0), \textsc{Ip}(-5), \epsilon, \epsilon, \textsc{Ip}(0)) \\ \textsc{Sub}(\textsc{Ip}(0), \textsc{Cp}(\texttt{.}, 1, 0), \epsilon, \epsilon, \textsc{Ip}(0)) \\ \textsc{Sub}(\textsc{Ip}(0), \textsc{Cp}(\texttt{.}, -1, 0), \epsilon, \epsilon, \textsc{Ip}(0)) \end{array} \right\}]$$

Synthesized rule

我们的工作：Shellfix

- Shellfix：基于语法的修复
  - 基本观察：错误的命令通常含有语法错误
  - 修复目标：将含有语法错误的命令修改为符合语法的命令
- 系统框架



DSL定义

Commands	*cmds*	:=	$cmd, cmds \mid cmd$
Command	*cmd*	:=	$\textsc{Command}(s, [args])$
Argument list	*args*	:=	$arg, args \mid arg \mid \epsilon$
Argument	*arg*	:=	$\textsc{Arg}(expr, mult)$
Argument expression	*expr*	:=	$\textsc{Str}(s) \mid \textsc{Identifier} \mid \textsc{Integer} \mid \textsc{Path} \mid \textsc{Glob} \mid \cdots$
Multiplicity	*mult*	:=	`'1'|'?'|'+'|'*'`
String	*s*	:=	arbitrary string

- 例：cp命令

Command(cp, [Arg(Str('-r'), ?), Arg(Str('-f'), ?), Arg(Path, +), Arg(Path, 1)])

是否递归拷贝文件夹

是否强制覆盖目标文件

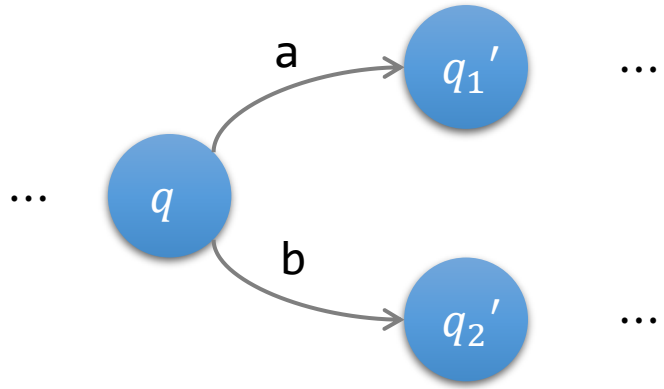
源文件列表

目标路径

Shellfix的修复算法

- 采用Levenshtein编辑距离作为修复代价的度量
- 问题：给定含有语法错误的命令s, 找出**满足语法要求**、且与s**编辑距离最小**的命令s'。
- 算法：给定NFA表示的语法$M(Q, \Sigma, \Delta, S, F)$和待修复命令$s$
  - 对于自动机状态q和位置a，节点(q, a)表示s的前a个字符匹配自动机状态q
  - 在对应的状态之间连边，边的权值表示对编辑距离的贡献
  - $(t, 0), t \in S$到$(f, |s|), f \in F$之间的最短路径即为所求的字符串

Shellfix的修复算法



NFA（部分）

foobar

待修复命令字符串

边的类型	示例
正确输入下一个字符	$(q, \text{foobar}) \xrightarrow{0} (q_2', \text{foobar})$
错误输入下一个字符	$(q, \text{foobar}) \xrightarrow{1} (q_1', \text{foobar})$
多输入一个字符	$(q, \text{foobar}) \xrightarrow{1} (q, \text{foobar})$
少输入一个字符	$(q, \text{foobar}) \xrightarrow{1} (q_1', \text{foobar})$

扩展和优化

Extension 1 (Top *k* extension)

- 求出编辑距离最小的k种修复方式，供用户选择

Extension 2 (Probability model extension)

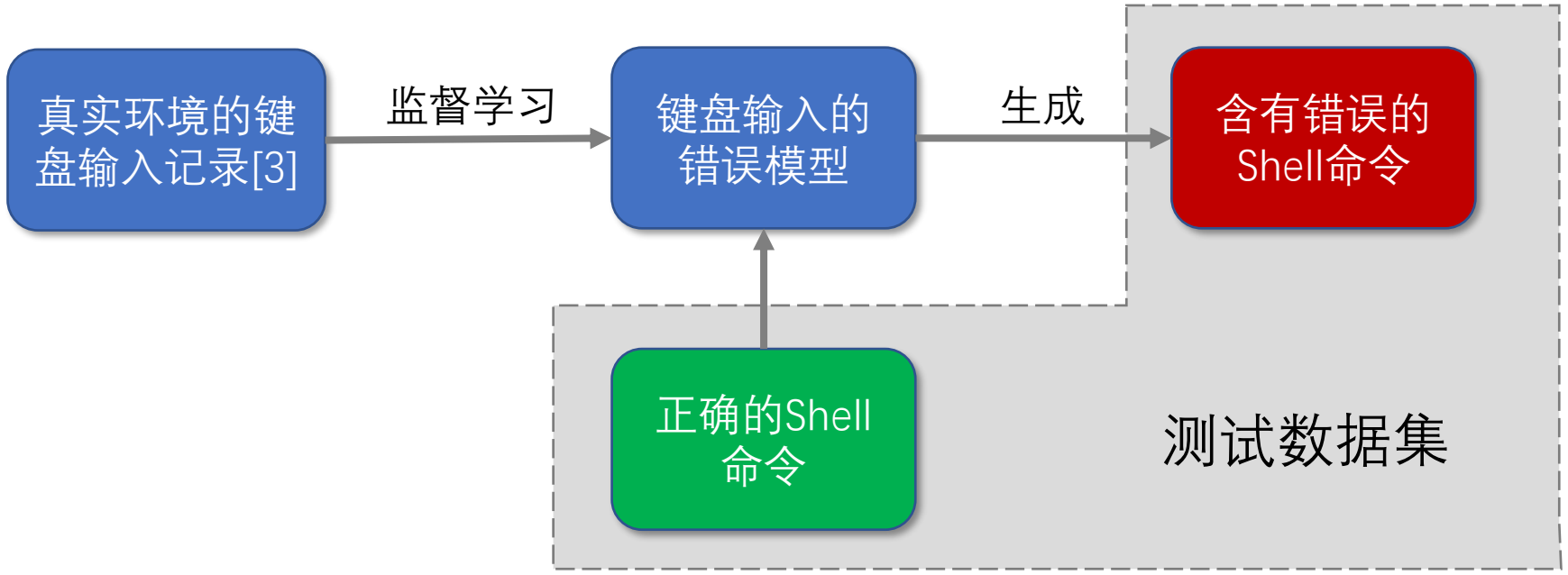
- 使用概率模型（而不是编辑距离）作为修复代价的度量标准

Extension 3 (Environmental oracle extension)

- 将语义信息嵌入到语法中，利用当前系统中的环境信息提高修复的成功率
- 例如：当命令的某个参数要求是系统中存在的文件时，修复后的命令该参数必须是系统中存在的文件名

评测数据集

- 目前，学术界关于Shell的研究工作较少，也没有合适的用于评测Shell命令修复工具的数据集。因此，我们自己构造了一套数据集，用于测试我们工具的修复效果：



系统实现

- Shellfix工具（包括DSL定义、解析以及修复算法）
  - 采用Python实现
  - 约800行代码
- 用于实验评估的数据集生成代码
  - 采用C++和Python实现
  - 约500行代码
- 用于评测Shell命令修复工具的数据集
  - 15个程序，36条正确命令，360条错误命令

实验评估

Program	# correct	# buggy	Example
apt-get	4	40	apt-get update
cat	2	20	cat myfile.txt
cd	2	20	cd sybase
chmod	4	40	chmod a-x file
echo	1	10	echo "Hello World"
expr	6	60	expr ss64 : ss6
kill	1	10	kill 1293
ln	4	40	ln file1.txt link1
ls	3	30	ls -al
man	3	30	man intro
mkdir	1	10	mkdir foo
mv	1	10	mv apple orange.doc
rmdir	1	10	rmdir myfolder
tail	1	10	tail -85 file.txt
touch	2	20	touch sample.txt
Σ	36	360	

	ShellFix		TheFxxx	
Program	#	%	#	%
apt-get	**31/40**	**77.50%**	10/40	25.00%
cat	**5/20**	**25.00%**	1/20	5.00%
cd	0/20	0.00%	0/20	0.00%
chmod	**17/40**	**42.50%**	15/40	37.50%
echo	**1/10**	**10.00%**	0/10	0.00%
expr	**13/60**	**21.67%**	10/60	16.67%
kill	3/10	30.00%	3/10	30.00%
ln	0/40	0.00%	0/40	0.00%
ls	0/30	0.00%	0/30	0.00%
man	1/30	3.33%	**5/30**	**16.67%**
mkdir	6/10	60.00%	6/10	60.00%
mv	0/10	0.00%	0/10	0.00%
rmdir	3/10	30.00%	**4/10**	**40.00%**
tail	1/10	10.00%	1/10	10.00%
touch	1/20	5.00%	**4/20**	**20.00%**
Σ	**82/360**	**22.78%**	59/360	16.39%

仅用语法信息就能取得较好的修复效果

需考虑语义和系统环境信息以取得更好的效果，将在以后的工作中改进

参考文献

1. Loris D’ Antoni, Rishabh Singh, and Michael Vaughn. NoFAQ: Synthesizing command repairs from examples. In *ESEC/FSE 2017,* New York, NY, USA.
2. Tessa A. Lau, Pedro Domingos, and Daniel S. Weld. Version space algebra and its application to programming by demonstration. *In ICML’ 00*, San Francisco, CA, USA.
3. Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. Observations on typing from 136 million keystrokes. In *CHI’ 18*, New York, NY, USA.

Q & A