Symbolic Execution of Maintainer Scripts

Nicolas Jeannerod and Ralf Treinen
joint work with Benedikt Becker, Claude Marché, Mihaela Sighireanu, Yann Régis-Gianas

IRIF, Université de Paris

July 21, 2019
Plan

1. Introduction
2. Symbolic Execution of Scripts
3. Symbolic Execution of Maintainer Scripts
4. Demo Time
5. Detected Bugs
6. Conclusions
<table>
<thead>
<tr>
<th></th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Symbolic Execution of Scripts</td>
</tr>
<tr>
<td>3</td>
<td>Symbolic Execution of Maintainer Scripts</td>
</tr>
<tr>
<td>4</td>
<td>Demo Time</td>
</tr>
<tr>
<td>5</td>
<td>Detected Bugs</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
The CoLiS project

- **Goal**: apply formal methods to the quality assessment of Debian maintainer scripts.
- Initial idea: use methods from formal *program verification*.
- Example of a use case: A `postrm` that deletes files from *unrelated packages*, see for instance Ralf’s talk at Debconf’16 for a concrete example.
- We only look at Posix shell scripts which are more than 99% of our maintainer scripts.
- We knew from the beginning that this is an ambitious goal: We will at best succeed partially.
The CoLiS project

- **Goal:** apply formal methods to the quality assessment of Debian maintainer scripts.

- **Initial idea:** use methods from formal *program verification*.

- Example of a use case: A *postrm* that deletes files from *unrelated packages*, see for instance Ralf’s talk at Debconf’16 for a concrete example.

- We only look at Posix shell scripts which are more than 99% of our maintainer scripts.

- We knew from the beginning that this is an ambitious goal: We will at best succeed partially.
The CoLiS project

- Goal: apply formal methods to the quality assessment of Debian maintainer scripts.
- Initial idea: use methods from formal program verification.
- Example of a use case: A postrm that deletes files from unrelated packages, see for instance Ralf’s talk at Debconf’16 for a concrete example.
- We only look at Posix shell scripts which are more than 99% of our maintainer scripts.
- We knew from the beginning that this is an ambitious goal: We will at best succeed partially.
The CoLiS project

- Goal: apply formal methods to the quality assessment of Debian maintainer scripts.
- Initial idea: use methods from formal program verification.
- Example of a use case: A postrm that deletes files from unrelated packages, see for instance Ralf’s talk at Debconf’16 for a concrete example.
- We only look at Posix shell scripts which are more than 99% of our maintainer scripts.
- We knew from the beginning that this is an ambitious goal: We will at best succeed partially.
The CoLiS project

- Goal: apply formal methods to the quality assessment of Debian maintainer scripts.
- Initial idea: use methods from formal program verification.
- Example of a use case: A postrm that deletes files from unrelated packages, see for instance Ralf’s talk at Debconf’16 for a concrete example.
- We only look at Posix shell scripts which are more than 99% of our maintainer scripts.
- We knew from the beginning that this is an ambitious goal: We will at best succeed partially.
What happened previously

- Static syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The Morbig parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- **Static** syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The *Morbig* parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- Static syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The Morbig parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- Static syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The Morbig parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- Static syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The Morbig parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- **Static** syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The *Morbig* parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- Static syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The Morbig parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What happened previously

- Static syntactic analysis of Posix shell scripts.
- Talks in 2018 at Fosdem, Minidebconf Hamburg, Debconf.
- Static syntactical analysis of Posix shell scripts is far from trivial.
- The Morbig parser for Posix shell scripts.
- First report of bugs on a relatively trivial level, like:
  - Missing strict mode
  - Wrong redirections
  - Wrong test expressions
What we will present today

- Analyzing the *behavior* of Maintainer Scripts
- Caveat 1: we will never be able to analyze all the \( > 30,000 \) maintainer scripts.
- Caveat 2: we have to cut corners in the model, and perform *approximations*.
- Focus on finding bugs (as opposed to guaranteeing correctness).
What we will present today

- Analyzing the *behavior* of Maintainer Scripts
- Caveat 1: we will never be able to analyze all the > 30,000 maintainer scripts.
- Caveat 2: we have to cut corners in the model, and perform approximations.
- Focus on finding bugs (as opposed to guaranteeing correctness).
What we will present today

- Analyzing the *behavior* of Maintainer Scripts
- Caveat 1: we will never be able to analyze all the > 30,000 maintainer scripts.
- Caveat 2: we have to cut corners in the model, and perform *approximations*.
- Focus on finding bugs (as opposed to guaranteeing correctness).
What we will present today

- Analyzing the *behavior* of Maintainer Scripts
- Caveat 1: we will never be able to analyze all the > 30,000 maintainer scripts.
- Caveat 2: we have to cut corners in the model, and perform *approximations*.
- Focus on finding bugs (as opposed to guaranteeing correctness).
Plan

1. Introduction
2. Symbolic Execution of Scripts
3. Symbolic Execution of Maintainer Scripts
4. Demo Time
5. Detected Bugs
6. Conclusions
Semantics of Shell Scripts

- First step: reasoning about one script at a time.
- Starting point: we need a language to talk about the semantics of scripts: symbolic representation.
- We do this both for the case of success and of failure of the script.
- We need a way to calculate effectively on these representations, and to combine them (sequential composition, conditional composition, …)
- Analogy: Using regular expressions to talk about sets of strings.
Semantics of Shell Scripts

- First step: reasoning about one script at a time.
- Starting point: we need a language to talk about the semantics of scripts: symbolic representation.
  - We do this both for the case of success and of failure of the script.
  - We need a way to calculate effectively on these representations, and to combine them (sequential composition, conditional composition, ...)
  - Analogy: Using regular expressions to talk about sets of strings.
Semantics of Shell Scripts

- First step: reasoning about one script at a time.
- Starting point: we need a language to talk about the semantics of scripts: symbolic representation.
- We do this both for the case of success and of failure of the script.
- We need a way to calculate effectively on these representations, and to combine them (sequential composition, conditional composition, ...)
- Analogy: Using regular expressions to talk about sets of strings.
Semantics of Shell Scripts

- First step: reasoning about one script at a time.
- Starting point: we need a language to talk about the semantics of scripts: symbolic representation.
- We do this both for the case of success and of failure of the script.
- We need a way to calculate effectively on these representations, and to combine them (sequential composition, conditional composition, ...)
- Analogy: Using regular expressions to talk about sets of strings.
Semantics of Shell Scripts

- First step: reasoning about one script at a time.
- Starting point: we need a language to talk about the semantics of scripts: symbolic representation.
- We do this both for the case of success and of failure of the script.
- We need a way to calculate effectively on these representations, and to combine them (sequential composition, conditional composition, …)
- Analogy: Using regular expressions to talk about sets of strings.
Tree Constraints

- Our current approach: use predicate logic.
- Predicate logic allows us to talk about relations: in our case the relation between the initial configuration, and the possible configurations obtained by the execution.
- Special purpose logic for talking about a restricted form of tree transformations.
- Effective calculations on formulas.
Tree Constraints

- Our current approach: use predicate logic.
- Predicate logic allows us to talk about *relations*: in our case the relation between the initial configuration, and the possible configurations obtained by the execution.
- Special purpose logic for talking about a restricted form of tree transformations.
- Effective calculations on formulas.
Tree Constraints

- Our current approach: use predicate logic.
- Predicate logic allows us to talk about *relations*: in our case the relation between the initial configuration, and the possible configurations obtained by the execution.
- Special purpose logic for talking about a restricted form of tree transformations.
- Effective calculations on formulas.
Tree Constraints

- Our current approach: use predicate logic.
- Predicate logic allows us to talk about relations: in our case the relation between the initial configuration, and the possible configurations obtained by the execution.
- Special purpose logic for talking about a restricted form of tree transformations.
- Effective calculations on formulas.
## Example Specification: \texttt{mkdir }q/f

| **Success** | \[ \exists x, x', y'. \]
|-------------|---------------------------------------------------------------|
|             | \[
|             | \text{resolve}(r, \text{cwd}, q, x) \land \text{dir}(x) \land x[f] \uparrow \]
|             | \[
|             | \land \text{similar}(r, r', \text{cwd}, q, x, x') \land x \sim \{f\} x' \]
|             | \[
|             | \land \text{dir}(x') \land x'[f] y' \land \text{dir}(y') \land y'[\emptyset] \]
| **Failure** | \textit{File exists}\[ \exists y \cdot \text{resolve}(r, \text{cwd}, q/f, y) \land r \doteq r' \]
| **Failure** | \textit{No such file}\[ \text{noresolve}(r, \text{cwd}, q) \land r \doteq r' \]
| **Failure** | \textit{Not a dir}\[ \exists x \cdot \text{resolve}(r, \text{cwd}, q, x) \land \neg \text{dir}(x) \land r \doteq r' \]
### Example Specification: `mkdir q/f`

| Success |  \( \exists x, x', y'. \)  
|---------|--------------------------------------------------|
|         | `\text{resolve}(r, cwd, q, x) \land \text{dir}(x) \land x[f] \uparrow`  
|         | `\land \text{similar}(r, r', cwd, q, x, x') \land x \sim \{f\} x'$  
|         | `\land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset]`  |
| Failure | **File exists**  
|         |  \( \exists y. \text{resolve}(r, cwd, q/f, y) \land r \not\doteq r' \)
| Failure | **No such file**  
|         | `\text{noresolve}(r, cwd, q) \land r \not\doteq r'$  |
| Failure | **Not a dir**  
<p>|         |  ( \exists x. \text{resolve}(r, cwd, q, x) \land \neg \text{dir}(x) \land r \not\doteq r'$  |</p>
<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text (for human beings)</th>
</tr>
</thead>
</table>
| Success                           | § \exists x, x', y' · resolve(r, cwd, q, x) \land \text{dir}(x) \land x[f] \uparrow \\
|                                  | \land \text{similar}(r, r', cwd, q, x, x') \land x \sim \{f\} \land x' \\
|                                  | \land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset] |
| Failure                           | File exists § \exists y \cdot resolve(r, cwd, q/f, y) \land r \doteq r' |
|                                  | No such file § noresolve(r, cwd, q) \land r \doteq r' |
| Failure                           | Not a dir § \exists x \cdot resolve(r, cwd, q, x) \land \neg \text{dir}(x) \land r \doteq r' |
## Example Specification: \texttt{mkdir q/f}

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text (for human beings)</th>
<th>Formula in our logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success</strong></td>
<td></td>
<td>(\exists x, x', y'.) (\text{resolve}(r, \text{cwd}, q, x) \land \text{dir}(x) \land x[f] \uparrow) (\land \text{similar}(r, r', \text{cwd}, q, x, x') \land x \sim {f} \ x') (\land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset])</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><em>File exists</em></td>
<td>(\exists y \cdot \text{resolve}(r, \text{cwd}, q/f, y) \land r \vdash r')</td>
</tr>
<tr>
<td></td>
<td><em>No such file</em></td>
<td>(\text{noresolve}(r, \text{cwd}, q) \land r \vdash r')</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><em>Not a dir</em></td>
<td>(\exists x \cdot \text{resolve}(r, \text{cwd}, q, x) \land \neg \text{dir}(x) \land r \vdash r')</td>
</tr>
</tbody>
</table>
### Example Specification: `mkdir q/f`

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text (for human beings)</th>
<th>Formula in our logic</th>
</tr>
</thead>
</table>
| Success                          |                                     | \[\exists x, x', y' . \]
|                                  |                                     | resolve\( (r, \text{cwd}, q, x) \land \text{dir}(x) \land x[f] \uparrow \land \text{similar}(r, r', \text{cwd}, q, x, x') \land x \sim \{f\} \land x' \land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset] \]
| Failure                          | *File exists*                       | \[\exists y . \]
|                                  |                                     | noresolve\( (r, \text{cwd}, q) \land r = r' \]
| Failure                          | *No such file*                      | \[\exists x . \]
|                                  |                                     | \text{notdir}(x) \land x \sim \{f\} \land x' \land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset] \]
| Failure                          | *Not a dir*                         | \[= r' \]

Nicolas Jeannerod, Ralf Treinen

IRIF, Université de Paris
### Example Specification: mkdir q/f

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text (for human beings)</th>
<th>Formula in our logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>∃x, x', y'. resolve(r, cwd, q, x) ∧ dir(x) ∧ x[f]↑ ∧ similar(r, r', cwd, q, x, x') ∧ x ∼ {f} x' ∧ dir(x') ∧ x'[f]y' ∧ dir(y') ∧ y'[∅]</td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>File exists</td>
<td>∃y. noreresolve(r, cwd, q, y)</td>
</tr>
<tr>
<td>Failure</td>
<td>No such file</td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>Not a dir</td>
<td></td>
</tr>
</tbody>
</table>

Nicolas Jeannerod, Ralf Treinen
Symbolic Execution of Maintainer Scripts
IRIF, Université de Paris
### Example Specification: `mkdir q/f`

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text</th>
<th>Formula in our logic</th>
</tr>
</thead>
</table>
| **Success**                       |                  | \( \exists x, x', y'. \)  
|                                  |                  | \( \text{resolve}(r, cwd, q, x) \land \text{dir}(x) \land x[f] \uparrow \)  
|                                  |                  | \( \land \text{similar}(r, r', cwd, q, x, x') \land x \sim \{f\} x' \)  
|                                  |                  | \( \land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset] \)  
| **Failure**                      | **File exists**  | \( \exists y. \)  
|                                  |                  | \( \text{resolve}(r, cwd, q/f, y) \land r = r' \)  
|                                  | **No such file** | \( \text{noresolve}(r, cwd, q) \land r = r' \)  
| **Failure**                      | **Not a dir**    | \( \exists x. \)  
|                                  |                  | \( \exists x \)  

Nicolas Jeannerod, Ralf Treinen  
IRIF, Université de Paris  
Symbolic Execution of Maintainer Scripts
**Example Specification:** `mkdir q/f`

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text</th>
<th>Formula in our logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success</strong></td>
<td></td>
<td>[\exists x, x', y' . resolve(r, cwd, q, x) \land dir(x) \land x[f] \uparrow \land similar(r, r', cwd, q, x, x') \land x \sim {f} x' \land dir(x') \land x'[f]y' \land dir(y') \land y'[\emptyset]]</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><strong>File exists</strong></td>
<td>[\exists y . \text{noresolve}(r, cwd, q/f, y) \land r = r']</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><strong>No such file</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><strong>Not a dir</strong></td>
<td>[\exists x . \text{resolve}(r, cwd, q, x) \land \neg dir(x) \land r = r']</td>
</tr>
</tbody>
</table>
Example Specification: \texttt{mkdir q/f}

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Formula in our logic</th>
</tr>
</thead>
</table>
| Success                         | \[\exists x, x', y'. \]
|                                 | resolve(r, cwd, q, x) \land \text{dir}(x) \land x[f] \uparrow \]
|                                 | \land \text{similar}(r, r', cwd, q, x, x') \land x \sim \{f\} x' \]
|                                 | \land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset] \]
| Failure                         | \exists y . \]
| File exists                     | \text{file exists} \]
| No such file                    | \text{noresolve}(r, cwd, q) \land r. = r' \]
| Not a dir                       | \exists x . \]
|                                 | (\text{dir}) \]
|                                 | \exists x' \]

Nicolas Jeannerod, Ralf Treinen
IRIF, Université de Paris
Example Specification: `mkdir q/f`

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Description Text (for human beings)</th>
<th>Formula in our logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success</strong></td>
<td><code>∃x, x', y'.</code></td>
<td>$\exists x, x', y'$.</td>
</tr>
<tr>
<td></td>
<td><code>resolve(r, cwd, q, x) \land dir(x) \land x[f]</code></td>
<td>$\exists x, x'$.</td>
</tr>
<tr>
<td></td>
<td>`\land similar(r, r', cwd, q, x, x') \land x \sim {f} x'$</td>
<td>$\land dir(x') \land x'[f] y' \land dir(y') \land y'[\emptyset]$</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><code>File exists</code></td>
<td>$\exists y'$.</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><code>No such file</code></td>
<td>$\exists x \cdot$</td>
</tr>
<tr>
<td></td>
<td><code>\noreolve(r, cwd, q/f, y) \land r . = r'</code></td>
<td>$\exists x \cdot$</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><code>Not a dir</code></td>
<td>$\exists x \cdot$</td>
</tr>
<tr>
<td></td>
<td><code>\nordi(r, cwd, q, x) \land \sim \{f\} \land x'</code></td>
<td>$\exists x'$.</td>
</tr>
</tbody>
</table>

Nicolas Jeannerod, Ralf Treinen
IRIF, Université de Paris

Symbolic Execution of Maintainer Scripts
Example Specification: \texttt{mkdir q/f}

<table>
<thead>
<tr>
<th>Success</th>
<th>Formula in our logic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\exists x, x', y' \cdot$</td>
</tr>
<tr>
<td></td>
<td>resolve(r, \textit{cwd}, q, x) \land \text{dir}(x) \land x[f] \uparrow$</td>
</tr>
<tr>
<td></td>
<td>$\land \text{similar}(r, r', \textit{cwd}, q, x, x') \land x \sim {q} x'$</td>
</tr>
<tr>
<td></td>
<td>$\land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset]$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Failure</th>
<th>Description Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>File exists</td>
<td>$\exists y \cdot$</td>
</tr>
<tr>
<td>No such file</td>
<td>$\exists x \cdot \text{resolve}(r, \textit{cwd}, q, x, y) \land r \neq r'$</td>
</tr>
<tr>
<td>Not a dir</td>
<td>$\exists x \cdot \text{resolve}(r, \textit{cwd}, q, x, y)$</td>
</tr>
</tbody>
</table>
### Example Specification: `mkdir q/f`

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Description</th>
<th>Formula in our logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success</strong></td>
<td></td>
<td>( \exists x, x', y'. )  \  ( \text{resolve}(r, \text{cwd}, q, x) \land \text{dir}(x) \land x[f] \uparrow ) \  ( \land \text{similar}(r, r', \text{cwd}, q, x, x') \land x \sim {f} x' ) \  ( \land \text{dir}(x') \land x'[f]y' \land \text{dir}(y') \land y'[\emptyset] )</td>
</tr>
<tr>
<td><strong>Failure</strong></td>
<td><em>File exists</em></td>
<td>( \exists y'. )  \  ( \text{notresolve}(r, \text{cwd}, q) \land r. = r' )</td>
</tr>
</tbody>
</table>
| **Failure**      | *No such file*             | \( \exists x. \)  \\
|                  |                             | \( \text{notexist}(r, \text{cwd}, q) \land r. = r' \) |
| **Failure**      | *Not a dir*                | \( \exists x. \)  \\
|                  |                             | \( \text{notdir}(r, \text{cwd}, q, x) \land r. = r' \) |

Nicolas Jeannerod, Ralf Treinen
IRIF, Université de Paris

Symbolic Execution of Maintainer Scripts
Example Specification: \texttt{mkdir \ q/f}

<table>
<thead>
<tr>
<th>Outcome of the Specification Case</th>
<th>Formula in our logic</th>
</tr>
</thead>
</table>
| **Success**                       | $\exists x, x', y' \cdot$
|                                   | $\text{resolve}(r, \text{cwd}, q, x) \land \text{dir}(x) \land x[f] \uparrow$
|                                   | $\land \text{similar}(r, r', \text{cwd}, q, x, x') \land x \sim \{f\} \ x'$
|                                   | $\land \text{dir}(x') \land x'[f] \downarrow y' \land \text{dir}(y') \land y'[\emptyset]$ |
| **Failure**                       | **File exists**      |
|                                   | $\exists y \cdot \text{resolve}(r, \text{cwd}, q/f, y) \land r \Downarrow r'$ |
| **Failure**                       | **No such file**     |
|                                   | $\text{noresolve}(r, \text{cwd}, q) \land r \Downarrow r'$ |
| **Failure**                       | **Not a dir**        |
|                                   | $\exists x \cdot \text{resolve}(r, \text{cwd}, q, x) \land \neg \text{dir}(x) \land r \Downarrow r'$ |
Using the Logic: sequential composition

\[ \exists \text{tmp}. (\text{cmd}_1(\text{in}, \text{tmp}) \land \text{cmd}_2(\text{tmp}, \text{out})) \]

\[ \text{cmd}_{1;2}(\text{in}, \text{out}) \]
Using the Logic: sequential composition

\[
\exists tmp. (cmd_1(in, tmp) \land cmd_2(tmp, out))
\]
Using the Logic: sequential composition

\[ \exists \text{tmp}. (\text{cmd}_1(\text{in}, \text{tmp}) \land \text{cmd}_2(\text{tmp}, \text{out})) \]
Symbolic Execution

- **Idea:** We simulate the script, and collect in our logical formalism its effect on the file system.

- More precisely: Mixed concrete/symbolic execution: We only describe symbolically the effect on the file system, other effects like variable assignments etc. are simulated concretely.

- We know the parameters the script is invoked on, and we make reasonable assumptions on environment variables.
Symbolic Execution

- Idea: We simulate the script, and collect in our logical formalism its effect on the file system.

- More precisely: Mixed concrete/symbolic execution: We only describe symbolically the effect on the file system, other effects like variable assignments etc. are simulated concretely.

- We know the parameters the script is invoked on, and we make reasonable assumptions on environment variables.
Symbolic Execution

- Idea: We simulate the script, and collect in our logical formalism its effect on the file system.

- More precisely: Mixed concrete/symbolic execution: We only describe symbolically the effect on the file system, other effects like variable assignments etc. are simulated concretely.

- We know the parameters the script is invoked on, and we make reasonable assumptions on environment variables.
Plan

1. Introduction
2. Symbolic Execution of Scripts
3. Symbolic Execution of Maintainer Scripts
4. Demo Time
5. Detected Bugs
6. Conclusions
Installation Scenarios

- Second Step: scenarios, like this one:
  Installation of foo (Not Installed)

- More (and more complex) scenarios: see the policy.
Installation Scenarios

- Second Step: scenarios, like this one:
  Installation of foo (Not Installed)

- More (and more complex) scenarios: see the policy.
Failures and bad states

- Three different kinds of observations:
  1. The failure (exit code > 0) of a maintainer script
  2. The failure of a request to dpkg
  3. The state a package is in at the end of the process

- As one can see in the scenarios:
  - It is possible that a request fails, but still all packages are in a consistent state: when the error unwind has worked.
  - There are situations where some script may fail, and still the request succeeds in the end.
Failures and bad states

- Three different kinds of observations:
  1. The failure (exit code > 0) of a maintainer script
  2. The failure of a request to dpkg
  3. The state a package is in at the end of the process

- As one can see in the scenarios:
  - it is possible that a request fails, but still all packages are in a consistent state: when the error unwind has worked.
  - there are situations where some script may fail, and still the request succeeds in the end.
Failures and bad states

- Three different kinds of observations:
  - The failure (exit code $> 0$) of a maintainer script
  - The failure of a request to dpkg
  - The state a package is in at the end of the process

- As one can see in the scenarios:
  - it is possible that a request fails, but still all packages are in a consistent state: when the error unwind has worked.
  - there are situations where some script may fail, and still the request succeeds in the end.
Failures and bad states

- Three different kinds of observations:
  1. The failure (exit code > 0) of a maintainer script
  2. The failure of a request to dpkg
  3. The state a package is in at the end of the process

- As one can see in the scenarios:
  - it is possible that a request fails, but still all packages are in a consistent state: when the *error unwind* has worked.
  - there are situations where some script may fail, and still the request succeeds in the end.
Failures and bad states

- Three different kinds of observations:
  1. The failure (exit code > 0) of a maintainer script
  2. The failure of a request to dpkg
  3. The state a package is in at the end of the process

- As one can see in the scenarios:
  - it is possible that a request fails, but still all packages are in a consistent state: when the error unwind has worked.
  - there are situations where some script may fail, and still the request succeeds in the end.
Failures and bad states

- Three different kinds of observations:
  1. The failure (exit code $> 0$) of a maintainer script
  2. The failure of a request to dpkg
  3. The state a package is in at the end of the process

- As one can see in the scenarios:
  - it is possible that a request fails, but still all packages are in a consistent state: when the error unwind has worked.
  - there are situations where some script may fail, and still the request succeeds in the end.
Failures and bad states

- Three different kinds of observations:
  1. The failure (exit code > 0) of a maintainer script
  2. The failure of a request to dpkg
  3. The state a package is in at the end of the process

- As one can see in the scenarios:
  - it is possible that a request fails, but still all packages are in a consistent state: when the error unwind has worked.
  - there are situations where some script may fail, and still the request succeeds in the end.
Failures and Bugs

- **Policy 6.1 says:**
  
  *The package management system looks at the exit status from these scripts. It is important that they exit with a non-zero status if there is an error, so that the package management system can stop its processing... It is also important, of course, that they exit with a zero status if everything went well.*

- **Consequence:** A possible failure case of a script is not necessarily a bug!
Failures and Bugs

- Policy 6.1 says:
  
  The package management system looks at the exit status from these scripts. It is important that they exit with a non-zero status if there is an error, so that the package management system can stop its processing... It is also important, of course, that they exit with a zero status if everything went well.

- Consequence: A possible failure case of a script is not necessarily a bug!
Failures and Bugs

- Policy 6.1 says:
  *The package management system looks at the exit status from these scripts. It is important that they exit with a non-zero status if there is an error, so that the package management system can stop its processing... It is also important, of course, that they exit with a zero status if everything went well.*

- Consequence: A possible failure case of a script is not necessarily a bug!
## Plan

1. Introduction
2. Symbolic Execution of Scripts
3. Symbolic Execution of Maintainer Scripts
4. Demo Time
5. Detected Bugs
6. Conclusions
Infrastructure

- Corpus of 13906 packages containing 33320 maintainer scripts extracted on 2019-03-18 from a Debian mirror
- Corpus of 165 additional files which are included by maintainer scripts
- Using the Contents file to simulate dpkg -L
- Running for 20 minutes on a 80 cores Intel(R) Xeon(R) CPU at 2.20GHz.
Infrastructure

- Corpus of 13906 packages containing 33320 maintainer scripts extracted on 2019-03-18 from a Debian mirror
- Corpus of 165 additional files which are included by maintainer scripts
  - Using the Contents file to simulate dpkg -L
  - Running for 20 minutes on a 80 cores Intel(R) Xeon(R) CPU at 2.20GHz.
Infrastructure

- Corpus of 13906 packages containing 33320 maintainer scripts extracted on 2019-03-18 from a Debian mirror
- Corpus of 165 additional files which are included by maintainer scripts
- Using the Contents file to simulate `dpkg -L`
- Running for 20 minutes on a 80 cores Intel(R) Xeon(R) CPU at 2.20GHz.
Infrastructure

- Corpus of 13906 packages containing 33320 maintainer scripts extracted on 2019-03-18 from a Debian mirror
- Corpus of 165 additional files which are included by maintainer scripts
- Using the Contents file to simulate dpkg -L
- Running for 20 minutes on a 80 cores Intel(R) Xeon(R) CPU at 2.20GHz.
Plan

1. Introduction
2. Symbolic Execution of Scripts
3. Symbolic Execution of Maintainer Scripts
4. Demo Time
5. Detected Bugs
6. Conclusions
sgml-base preinst

- Script snippet:

```bash
if [ ! -d /var/lib/sgml-base ]
then
    mkdir /var/lib/sgml-base 2>/dev/null
fi
```

- Problem: If `/var/lib/sgml-base` exists and is not a directory, this fails *silently*.

- We have asked on the mailing list for confirmation that this is a bug.

- [https://bugs.debian.org/929706](https://bugs.debian.org/929706)
**sgml-base preinst**

- **Script snippet:**

  ```bash
  if [ ! -d /var/lib/sgml-base ]
  then
    mkdir /var/lib/sgml-base 2>/dev/null
  fi
  ```

- **Problem:** If `/var/lib/sgml-base` exists and is not a directory this fails *silently*

- We have asked on the mailing list for confirmation that this is a bug.

- [https://bugs.debian.org/929706](https://bugs.debian.org/929706)
sgml-base preinst

- Script snippet:

  ```bash
  if [ ! -d /var/lib/sgml-base ]
  then
    mkdir /var/lib/sgml-base 2>/dev/null
  fi
  
  Problem: If /var/lib/sgml-base exists and is not a directory this fails *silently*
  
  We have asked on the mailing list for confirmation that this is a bug.

  [https://bugs.debian.org/929706](https://bugs.debian.org/929706)
Script snippet:

```bash
if [ ! -d /var/lib/sgml-base ]
then
    mkdir /var/lib/sgml-base 2>/dev/null
fi
```

Problem: If `/var/lib/sgml-base` exists and is not a directory this fails *silently*

We have asked on the mailing list for confirmation that this is a bug.

[https://bugs.debian.org/929706](https://bugs.debian.org/929706)
armagetronad-dedicated postrm

- **Script snippet:**

  ```bash
  if [ "$1" = "purge" ]; then
    rm -r /var/games/armagetronad
    rmdir --ignore-fail-on-non-empty /var/games
  fi
  ```

  - Will fail if `/var/games/armagedtrontad` does not exist.
  - Do we have to account for this case?
  - Policy, section 6.2: Maintainer scripts have to be idempotent.
  - Note that if a `postrm purge` succeeds the package is gone completely.
  - We still think this is a bug since the script may fail later.
armagetronad-dedicated postrm

- **Script snippet:**
  ```bash
  if [ "$1" = "purge" ]; then
      rm -r /var/games/armagetronad
      rmdir --ignore-fail-on-non-empty /var/games
  fi
  ```

- **Will fail if** /var/games/armagedtronad **does not exist.**

- Do we have to account for this case?

- Policy, section 6.2: Maintainer scripts have to be idempotent.

- Note that if a `postrm purge` succeeds the package is gone completely.

- We still think this is a bug since the script may fail later.
armagetronad-dedicated postrm

- **Script snippet:**

  ```sh
  if [ "$1" = "purge" ]; then
    rm -r /var/games/armagetronad
    rmdir --ignore-fail-on-non-empty /var/games
  fi
  ```

- **Will fail if** /var/games/armagedtronad **does not exist.**

- **Do we have to account for this case?**

  - Policy, section 6.2: Maintainer scripts have to be idempotent.
  - Note that if a `postrm purge` succeeds the package is gone completely.
  - We still think this is a bug since the script may fail later.
armagetronad-dedicated postrm

- Script snippet:

  ```bash
  if [ "$1" = "purge" ]; then
    rm -r /var/games/armagetronad
    rmdir --ignore-fail-on-non-empty /var/games
  fi
  ```

- Will fail if `/var/games/armagedtronad` does not exist.
- Do we have to account for this case?
- Policy, section 6.2: Maintainer scripts have to be idempotent.
- Note that if a `postrm purge` succeeds the package is gone completely.
- We still think this is a bug since the script may fail later.
armagetronad-dedicated postrm

- Script snippet:

```bash
if [ "$1" = "purge" ]; then
    rm -r /var/games/armagetronad
    rmdir --ignore-fail-on-non-empty /var/games
fi
```

- Will fail if `/var/games/armagedtronad` does not exist.
- Do we have to account for this case?
- Policy, section 6.2: Maintainer scripts have to be idempotent.
- Note that if a `postrm purge` succeeds the package is gone completely.
- We still think this is a bug since the script may fail later.
armagetronad-dedicated postrm

- Script snippet:

  ```bash
  if [ "$1" = "purge" ]; then
    rm -r /var/games/armagetronad
    rmdir --ignore-fail-on-non-empty /var/games
  fi
  ```

- Will fail if /var/games/armagedtronad does not exist.

- Do we have to account for this case?

- Policy, section 6.2: Maintainer scripts have to be idempotent.

- Note that if a postrm purge succeeds the package is gone completely.

- We still think this is a bug since the script may fail later.
Idempotency

- Debian policy (section 6.2) requires maintainer scripts to be idempotent.

- Mathematically, $i$ is idempotent when

$$ i \circ i = i $$

- The sense in Debian is much larger: 
  If the first call failed, or aborted half way through for some reason, the second call should merely do the things that were left undone the first time, if any, and exit with a success status if everything is OK.
Idempotency

- Debian policy (section 6.2) requires maintainer scripts to be idempotent.
- Mathematically, $i$ is idempotent when
  \[ i \circ i = i \]
- The sense in Debian is much larger:
  If the first call failed, or aborted half way through for some reason, the second call should merely do the things that were left undone the first time, if any, and exit with a success status if everything is OK.
Idempotency

- Debian policy (section 6.2) requires maintainer scripts to be idempotent.
- Mathematically, $i$ is idempotent when
  \[ i \circ i = i \]
- The sense in Debian is much larger:
  
  * If the first call failed, or aborted half way through for some reason, the second call should merely do the things that were left undone the first time, if any, and exit with a success status if everything is OK."
Idempotency

- Debian policy (section 6.2) requires maintainer scripts to be idempotent.
- Mathematically, \( i \) is *idempotent* when
  \[
  i \circ i = i
  \]
- The sense in Debian is much larger: *If the first call failed, or aborted half way through for some reason, the second call should merely do the things that were left undone the first time, if any, and exit with a success status if everything is OK.*
courier-filter-perl postrm

- **Script snippet:**

```perl
case "$1" in
  purge )
    rm /etc/courier/filters/courier-filter-perl.conf
  ;;
esac
```

- Will fail when .../courier-filter-perl.conf does not exist: script not idempotent.

- However, this is at the end of script, so when it succeeds and removes the file the package is gone, so this seems purely formal.
courier-filter-perl postrm

- **Script snippet:**

```python
    case "$1" in
      purge )
        rm /etc/courier/filters/courier-filter-perl.conf
      ;;
    esac
```

- **Will fail when** ...
  /courier-filter-perl.conf does not exist:
  script not idempotent.

- However, this is at the end of script, so when it succeeds and removes the file the package is gone, so this seems purely formal.
courier-filter-perl postrm

- Script snippet:
  ```bash
  case "$1" in
    purge )
      rm /etc/courier/filters/courier-filter-perl.conf
      ;;
  esac
  Will fail when ...
courier-filter-perl.conf does not exist:
script not idempotent.
- However, this is at the end of script, so when it succeeds and
removes the file the package is gone, so this seems purely
formal.
oz postrm

- **Script snippet:**

```bash
FILE="/etc/oz/id_rsa-icicle-gen"

```case "$1" in
    purge)
        if [ -f $FILE ]; then
            rm $FILE $FILE.pub
        fi
    ;;
esac
```

- Fails if $FILE exists but $FILE.pub does not.
- In that case, a second invocation of `postrm purge` will succeed!
- Even if it is not against idempotency, this behavior is at least strange and annoying.
oz postrm

- **Script snippet:**

```bash
FILE="/etc/oz/id_rsa-icicle-gen"
case "$1" in
    purge)
        if [ -f $FILE ]; then
            rm $FILE $FILE.pub
        fi
    ;;
esac
```

- **Fails if $FILE exists but $FILE.pub does not.**
  - In that case, a second invocation of `postrm purge` will succeed!
  - Even if it is not against idempotency, this behavior is at least strange and annoying.

Nicolas Jeannerod, Ralf Treinen
IRIF, Université de Paris
oz postrm

- Script snippet:

```bash
FILE="/etc/oz/id_rsa-icicle-gen"

case "$1" in
  purge)
    if [ -f $FILE ]; then
      rm $FILE $FILE.pub
    fi
  ;;
esac
```

- Fails if $FILE exists but $FILE.pub does not.
- In that case, a second invocation of `postrm purge` will succeed!
- Even if it is not against idempotency, this behavior is at least strange and annoying.
oz postrm

- **Script snippet:**

```bash
FILE="/etc/oz/id_rsa-icicle-gen"
case "$1" in
  purge)
    if [ -f $FILE ]; then
      rm $FILE $FILE.pub
    fi
  ;;
esac

- Fails if $FILE exists but $FILE.pub does not.
- In that case, a second invocation of postrm purge will succeed!
- Even if it is not against idempotency, this behavior is at least strange and annoying.

Nicolas Jeannerod, Ralf Treinen
IRIF, Université de Paris
Bugs found by Colis

- Listing: https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=colis-shparser;users=treinen@debian.org

- 148 bugs filed so far, 90 of which are solved.

- So far a great majority are on a trivial level (like missing `set -e`), or on the level of syntactic structure (requires `morbig`, hence is not trivial).

- How did we find the last four bugs:
  - The first two from bad package states detected by our tool, then investigation by hand.
  - The last two where found by running our tool on a dedicated scenario for testing a subcase of idempotency.
Bugs found by Colis

- Listing: https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=colis-shparser;users=treinen@debian.org
- 148 bugs filed so far, 90 of which are solved.
- So far a great majority are on a trivial level (like missing `set -e`), or on the level of syntactic structure (requires `morbig`, hence is not trivial).
- How did we find the last four bugs:
  - The first two from bad package states detected by our tool, then investigation by hand.
  - The last two where found by running our tool on a dedicated scenario for testing a subcase of idempotency.
Bugs found by Colis

- Listing: https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=colis-shparser;users=treinen@debian.org
- 148 bugs filed so far, 90 of which are solved.
- So far a great majority are on a trivial level (like missing `set -e`), or on the level of syntactic structure (requires `morbig`, hence is not trivial).
- How did we find the last four bugs:
  - The first two from bad package states detected by our tool, then investigation by hand.
  - The last two where found by running our tool on a dedicated scenario for testing a subcase of idempotency.
Bugs found by Colis

- Listing: https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=colis-shparser;users=treinen@debian.org
- 148 bugs filed so far, 90 of which are solved.
- So far a great majority are on a trivial level (like missing `set -e`), or on the level of syntactic structure (requires `morbig`, hence is not trivial).
- How did we find the last four bugs:
  - The first two from bad package states detected by our tool, then investigation by hand.
  - The last two where found by running our tool on a dedicated scenario for testing a subcase of idempotency.
Bugs found by Colis

- Listing: https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=colis-shparser;users=treinen@debian.org
- 148 bugs filed so far, 90 of which are solved.
- So far a great majority are on a trivial level (like missing `set -e`), or on the level of syntactic structure (requires `morbig`, hence is not trivial).
- How did we find the last four bugs:
  - The first two from bad package states detected by our tool, then investigation by hand.
  - The last two where found by running our tool on a dedicated scenario for testing a subcase of idempotency.
Bugs found by Colis

- Listing: https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=colis-shparser;users=treinen@debian.org
- 148 bugs filed so far, 90 of which are solved.
- So far a great majority are on a trivial level (like missing `set -e`), or on the level of syntactic structure (requires `morbig`, hence is not trivial).
- How did we find the last four bugs:
  - The first two from bad package states detected by our tool, then investigation by hand.
  - The last two were found by running our tool on a dedicated scenario for testing a subcase of idempotency.
Plan

1. Introduction
2. Symbolic Execution of Scripts
3. Symbolic Execution of Maintainer Scripts
4. Demo Time
5. Detected Bugs
6. Conclusions
Ongoing Work

- Include simulation of the *unpack* phase.
- Increase the number of script we can handle, by modeling more commands.
- Being more precise about idempotency: checking *equivalence* of the executing a script once or twice.
- This uses our result on *decidability* of the logic.
- Investigate other properties, like commutation of scripts.
- Using *tree transducers* to represent the semantics of scripts.
Ongoing Work

- Include simulation of the *unpack* phase.
- Increase the number of script we can handle, by modeling more commands.
  - Being more precise about idempotency: checking *equivalence* of the executing a script once or twice.
  - This uses our result on *decidability* of the logic.
- Investigate other properties, like commutation of scripts.
- Using *tree transducers* to represent the semantics of scripts.
Ongoing Work

- Include simulation of the *unpack* phase.
- Increase the number of script we can handle, by modeling more commands.
- Being more precise about idempotency: checking *equivalence* of the executing a script once or twice.
- This uses our result on *decidability* of the logic.
- Investigate other properties, like commutation of scripts.
- Using *tree transducers* to represent the semantics of scripts.
Ongoing Work

- Include simulation of the *unpack* phase.
- Increase the number of script we can handle, by modeling more commands.
- Being more precise about idempotency: checking *equivalence* of the executing a script once or twice.
- This uses our result on *decidability* of the logic.
- Investigate other properties, like commutation of scripts.
- Using *tree transducers* to represent the semantics of scripts.
Ongoing Work

- Include simulation of the *unpack* phase.
- Increase the number of script we can handle, by modeling more commands.
- Being more precise about idempotency: checking *equivalence* of the executing a script once or twice.
- This uses our result on *decidability* of the logic.
- Investigate other properties, like commutation of scripts.
- Using *tree transducers* to represent the semantics of scripts.
Ongoing Work

- Include simulation of the *unpack* phase.
- Increase the number of script we can handle, by modeling more commands.
- Being more precise about idempotency: checking equivalence of the executing a script once or twice.
- This uses our result on *decidability* of the logic.
- Investigate other properties, like commutation of scripts.
- Using *tree transducers* to represent the semantics of scripts.
Thank you

- Joint work with the people from the Colis project.
- Project ANR-15-CE25-0001 funded by Agence Nationale de Recherche.
- October 2015 – September 2020
- http://colis.irif.fr/
Academic Papers

- NJ, CM, RT: *A Formally Verified Interpreter for a Shell-like Programming Language*, VSTTE 2017, https://hal.archives-ouvertes.fr/hal-01534747
- NJ, RT: *Deciding the First-Order Theory of an Algebra of Feature Trees with Updates*, IJCAR 2018, https://hal.archives-ouvertes.fr/hal-01807474
dpkg-maintscript-helper

- This is a utility that may be used by maintainer scripts
  - Script snippet:
    ```bash
    find "$PATHNAME" -mindepth 1 -print0 | \ 
    xargs -0 -i% mv -f "%" "$ABS_SYMLINK_TARGET/"
    ```
  - Fails when "$PATHNAME" contains subdirectories
  - Solution: add option "-maxdepth 1" to find
  - [https://bugs.debian.org/922799](https://bugs.debian.org/922799) (our proposed fix was accepted)
dpkg-maintscript-helper

- This is a utility that may be used by maintainer scripts
- Script snippet:

```
find "PATHNAME" -mindepth 1 -print0 | \
    xargs -0 -i% mv -f "%" "$ABS_SYMLINK_TARGET/"
```

- Fails when "PATHNAME" contains subdirectories
- Solution: add option "-maxdepth 1" to find
- https://bugs.debian.org/922799 (our proposed fix was accepted)
dpkg-maintscript-helper

- This is a utility that may be used by maintainer scripts
- Script snippet:
  ```bash
  find "$PATHNAME" -mindepth 1 -print0 | \n  xargs -0 -i% mv -f "%" "$ABS_SYMLINK_TARGET/"
  ```
- Fails when "$PATHNAME" contains subdirectories
  - Solution: add option "-maxdepth 1" to find
- [https://bugs.debian.org/922799](https://bugs.debian.org/922799) (our proposed fix was accepted)
dpkg-maintscript-helper

- This is a utility that may be used by maintainer scripts
- Script snippet:
  ```
  find "\$PATHNAME" -mindepth 1 -print0 | \n  xargs -0 -i% mv -f "%" "\$ABS_SYMLINK_TARGET/"
  ```
- Fails when "\$PATHNAME" contains subdirectories
- Solution: add option "-maxdepth 1" to find
- [https://bugs.debian.org/922799](https://bugs.debian.org/922799) (our proposed fix was accepted)
dpkg-maintscript-helper

- This is a utility that may be used by maintainer scripts
- Script snippet:
  ```bash
  find "\$PATHNAME" -mindepth 1 -print0 | \
  xargs -0 -i\% mv -f "\%" "\$ABS_SYMLINK_TARGET/"
  ```
- Fails when "\$PATHNAME" contains subdirectories
- Solution: add option "-maxdepth 1" to find
- [https://bugs.debian.org/922799](https://bugs.debian.org/922799) (our proposed fix was accepted)
Scenario: fresh installation

Installation of foo (Not Installed)

- preinst install: OK → Files are unpacked
- postinst configure: FAILED → "Installed"
- postrm abort-install: FAILED → "Failed-Config" → "Half Installed" → "Reinst required"
- Successful exit
- Exit with error message
Scenario: installation of previously removed package

```
Installation of foo 1.2-4 (Config-Files 1.2-3)
```

1. preinst install 1.2-3 → OK → Files are unpacked
2. postinst configure 1.2-3 → FAILED → "Installed"
3. postrm abort-install 1.2-3 → OK → "Config Files"
4. "Failed-Config" → "Half Installed" → Exit with error message
5. Successful exit
Scenario: upgrade of an installed package
Scenario: removal of an installed package

Removal of foo (Installed)

1. `prerm remove` → FAILED
2. `postrm remove` → FAILED
   - OK: Files are deleted
3. `postinst abort-remove` → FAILED
   - OK: "Installed"
   - "Half-Installed"
4. "Config-Files"
5. Successful exit
6. Exit with error message
Scenario: purge of a removed package

Purge of foo (Config-Files)

- `Conffiles are deleted`
- `Filelist is removed`
- "Not Installed"

- `postrm purge` (FAILED)
- Successful exit

- Exit with error message

Nicolas Jeannerod, Ralf Treinen
IRIF, Université de Paris
Scenario: purge of an installed package