# Le coquillage dans le CoLiS-mateur

## Formalizing a shell-like programming language

Nicolas Jeannerod

5th january 2017

# The CoLiS project

- Correctness of Linux Scripts
- Five years (october 2015 – september 2020)

# The CoLiS project

- Correctness of Linux Scripts
- Five years (october 2015 – september 2020)

- Goal: Apply verification techniques to shell scripts. . .

# The CoLiS project

- Correctness of Linux Scripts
- Five years (october 2015 – september 2020)

- Goal: Apply verification techniques to shell scripts. . .
  . . . in the Debian packages

# The CoLiS project

- Correctness of Linux Scripts
- Five years (october 2015 – september 2020)

- Goal: Apply verification techniques to shell scripts. . .
  . . . in the Debian packages

- Specific use case
- Uniform scripts corpus
- Debian Policy

# Example

```
#!/bin/sh
set -e

if [ ! -e /usr/local/lib/ocaml/4.01.0/stublibs ]; then
  if mkdir /usr/local/lib/ocaml/4.01.0/stublibs 2>/dev/null; th
    chown root:staff /usr/local/lib/ocaml/4.01.0/stublibs
    chmod 2775 /usr/local/lib/ocaml/4.01.0/stublibs
  fi
fi

for i in /usr/lib/ocaml/3.06 /etc/ocaml /var/lib/ocaml
do
  if [ -e \$i/ld.conf ]; then
    echo "Removing leftover $i/ld.conf"
    rm -f $i/ld.conf
    rmdir --ignore-fail-on-non-empty $i
```

# Motivation

- Everywhere in the world;
- Important administration tasks:
  - initialization,
  - installation, upgrade, removal of packages,
  - repetitive tasks (crontabs);

# Motivation

- Everywhere in the world;
- Important administration tasks:
  - initialization,
  - installation, upgrade, removal of packages,
  - repetitive tasks (crontabs);

  . . . ran as *root* user.

# Motivation

- Everywhere in the world;
- Important administration tasks:
  - initialization,
  - installation, upgrade, removal of packages,
  - repetitive tasks (crontabs);

  ... ran as *root* user.

- Treaterous syntax and semantics:
  Mistakes can happen pretty fast and be deadly:
  - `rm -rf /usr /lib/nvidia-current/xorg/xorg`,
  - `rm -rf "$STEAMROOT"/*` when `$STEAMROOT` is empty.

# Execute arbitrary strings

- Execute commands from strings:

```
a="echo foo"
$a        # echoes "foo"
"$a"      # fails
```

# Execute arbitrary strings

- Execute commands from strings:

```
a="echo foo"
$a        # echoes "foo"
"$a"      # fails
```

- . . . or any code with eval:

```
eval "if true; then echo foo; fi"
```

# Dynamic

- Everything is dynamic:

```
f () { g; }
g () { a=bar; }
a=foo
f
echo $a      ## echoes "bar"
```

# Dynamic

- Everything is dynamic:

```
f () { g; }
g () { a=bar; }
a=foo
f
echo $a      ## echoes "bar"
```

- Example 2-in-1 (expansion and dynamic scoping):

```
f () { echo $1 $a; }
a=foo
a=bar f $a      ## echoes "foo bar"
echo $a         ## echoes "bar"
```

# Behaviours

- Nice falses and the violent one:

```
false && true
! true
false
```

# Behaviours

- Nice falses and the violent one:

```
false && true
! true
false
```

-

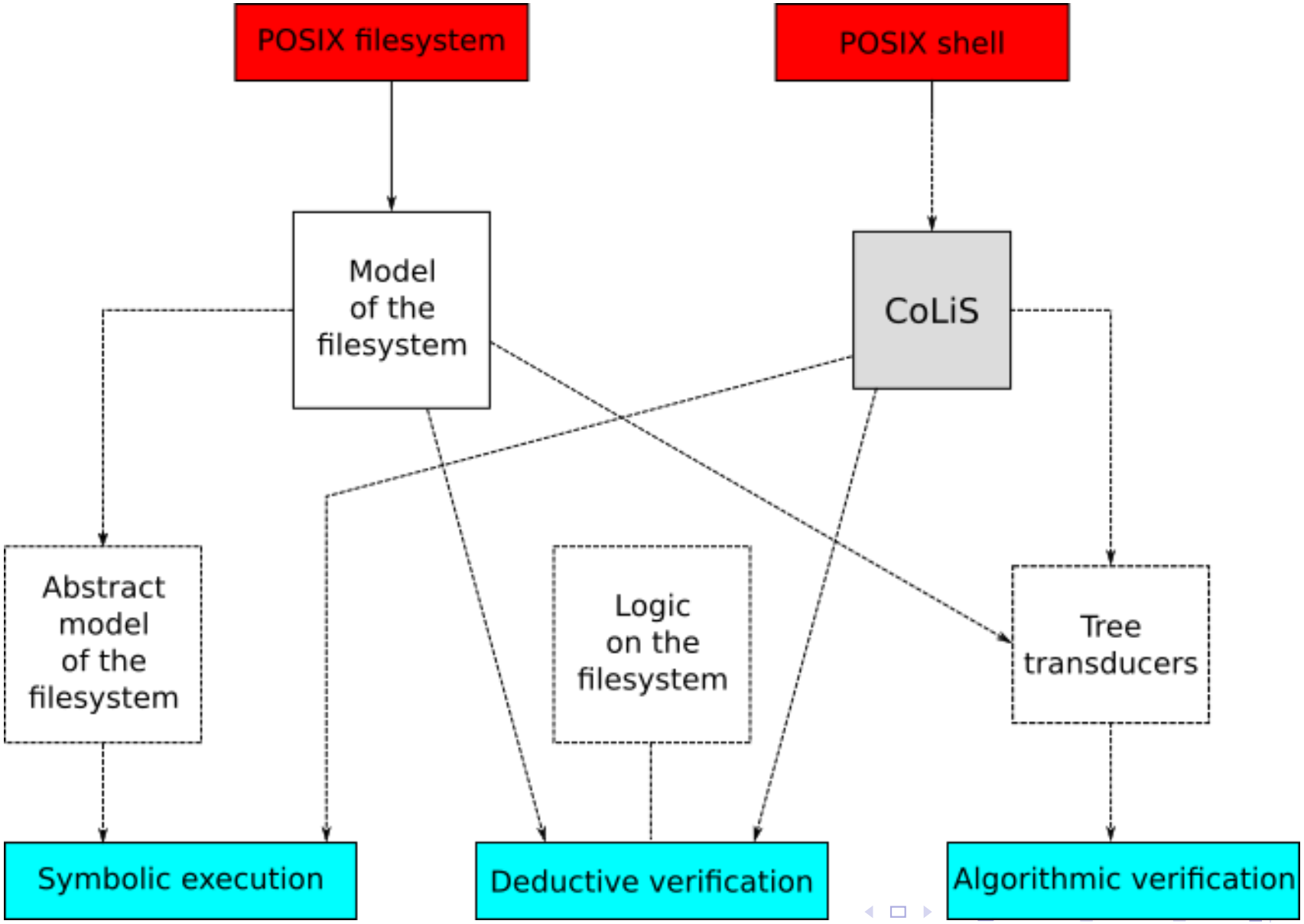
```
( exit )
( return )
exit | true
exit
```

# Hope

In our use case, scripts are quite simple.

- no recursion,
- only finite loops,
- no arithmetic,
- we don't distinguish between non-nul return codes,
- there's no concurrency:
  - neither external,
  - nor internal with &,
  - nor internal with |.

# Plan


POSIX filesystem
POSIX shell
Model of the filesystem
CoLiS
Abstract model of the filesystem
Logic on the filesystem
Tree transducers
Symbolic execution
Deductive verification
Algorithmic verification

# Requirements

- Close to a subset of shell;
- "Cleaner" than shell:
  - well defined syntax and semantics,
  - remove dangerous structures,
  - restrict the expansion mechanism;
- Must cover a good part of the scripts
  Checked by running statistics on a corpus of scripts from Debian;
- Keep the compilation shell $\rightarrow$ CoLiS feasible.

# Problems encountered

- The POSIX standard
- The expansion mechanism:
  - Has a huge expression power
  - Uses strings to encode integers
  - Uses strings to encode lists:

```
A="-l -a"
A="$A -h"
B="/home"
B="$B/nicolas"
ls $A $B
```

# Syntax

| | | | |
|---|---|---|---|
| String variables | $x_s$ | $\in$ | $SVar$ |
| List variables | $x_l$ | $\in$ | $LVar$ |
| Natural numbers | $n$ | $\in$ | $\mathbb{N}$ |
| Strings | $\sigma$ | $\in$ | $String$ |

| | | | |
|---|---|---|---|
| String expressions | $s$ | $::=$ | $f_s^*$ |
| String fragments | $f_s$ | $::=$ | $\sigma \mid x_s \mid n \mid t$ |
| List expressions | $l$ | $::=$ | $f_l^*$ |
| List fragments | $f_l$ | $::=$ | $[s] \mid \textbf{split}\ s \mid x_l$ |

| | | | |
|---|---|---|---|
| Terms | $t$ | $::=$ | $\textbf{true} \mid \textbf{false} \mid \textbf{fatal}$ |
| | | $\mid$ | $\textbf{return}\ t \mid \textbf{exit}\ t$ |
| | | $\mid$ | $x_s := s \mid x_l := l$ |
| | | | $\ldots$ |

# Semantic judgements

| | | | |
|---|---|---|---|
| Strings | $\sigma$ | $\in$ | $String$ |
| Lists | $\lambda$ | $\in$ | $StringList \triangleq \{ \sigma^* \mid \sigma \in \mathsf{String} \}$ |
| Term behaviours | $b$ | $\in$ | $\{\mathsf{True}, \mathsf{False}, \mathsf{Fatal}, \mathsf{Return\ True}$ |
| | | | $\mathsf{Return\ False}, \mathsf{Exit\ True}, \mathsf{Exit\ False}\}$ |
| Contexts | $\Gamma$ | $\in$ | $\mathcal{FS} \times \mathsf{String} \times \mathsf{StringList} \times \mathsf{SEnv} \times \mathsf{LEnv}$ |

| | | | |
|---|---|---|---|
| Terms | $t_{/\Gamma}$ | $\Rightarrow$ | $\sigma \star b_{/\Gamma'}$ |
| String fragments | $f_{s\,/\Gamma}$ | $\rightsquigarrow_{sf}$ | $\sigma \star \beta_{/\Gamma'}$ |
| String expressions | $s_{/\Gamma}$ | $\rightsquigarrow_{s}$ | $\sigma \star \beta_{/\Gamma'}$ |
| List fragments | $f_{l\,/\Gamma}$ | $\rightsquigarrow_{lf}$ | $\lambda \star \beta_{/\Gamma'}$ |
| List expressions | $l_{/\Gamma}$ | $\rightsquigarrow_{l}$ | $\lambda \star \beta_{/\Gamma'}$ |

# Formalization in Why3

- About Why3:
  - Platform for deductive program verification,
  - Relies on external provers,
  - Standard library,
  - WhyML: specification and programming language;

# Formalization in Why3

- About Why3:
  - Platform for deductive program verification,
  - Relies on external provers,
  - Standard library,
  - WhyML: specification and programming language;

- About CoLiS:
  - The syntax is a set of types *à la* OCaml;
  - The semantics are a set of inductive predicates *à la* Coq;
  - The interpreter is a set of WhyML functions
    ... proven correct *w.r.t* the semantics

# Ongoing work

- Rewrite our statistics using something more precise than `grep`;
- A model for the filesystem
- A compiler from shell to CoLiS:
  - In OCaml,
  - We already have a POSIX parser,
  - It will need to analyse;
- A test procedure for the compiler.

# Thanks for your attention

- Correctness of Linux Scripts
- ANR project ANR-15-CE25-0001
- `http://colis.irif.fr`
- Three teams:
  - IRIF (Paris Diderot)
  - INRIA Paris Saclay
  - INRIA Lille

- `https://nicolas.jeannerod.fr/research/le-coquillage-dans-le-colis-mateur`
- Questions?

# Informal semantics of the expansion

$$\frac{l \rightsquigarrow l' \qquad explode(IFS, l') = [a_0, a_1, \ldots a_n] \qquad a_0([a_1, \ldots a_n]) \rightarrow o}{l \Rightarrow o}$$

# Informal semantics of the expansion

$$\frac{l \rightsquigarrow l' \qquad explode(IFS, l') = [a_0, a_1, \ldots a_n] \qquad a_0([a_1, \ldots a_n]) \rightarrow o}{l \Rightarrow o}$$

$$\frac{}{\text{“...”} \rightsquigarrow \text{“...”}} \qquad \frac{}{\$v \rightsquigarrow E[\$v]} \qquad \frac{t \Rightarrow o}{\$(t) \rightsquigarrow o}$$

# Informal semantics of the expansion

$$\frac{l \rightsquigarrow l' \qquad explode(IFS, l') = [a_0, a_1, \ldots a_n] \qquad a_0([a_1, \ldots a_n]) \rightarrow o}{l \Rightarrow o}$$

$$\frac{}{\text{“...”} \rightsquigarrow \text{“...”}} \qquad \frac{}{\$v \rightsquigarrow E[\$v]} \qquad \frac{t \Rightarrow o}{\$(t) \rightsquigarrow o}$$

$$\frac{l \rightsquigarrow l' \qquad explode(IFS, l') = []}{l \Rightarrow ""}$$

# Examples

- Arithmetic:

```
$((expression))
```

Casts from strings to integers back to strings everywhere.

- Commands from strings:

```
a="echo foo"
$a        # echoes "foo"
"$a"      # fails
```

- or any code with eval:

```
eval "if true; then echo foo; fi"
```

# Examples – 2

- Everything is dynamic

```
f () { g; }
g () { a=bar; }
a=foo
f
echo $a      ## echoes "bar"
```

# Examples – 2

- Everything is dynamic

```
f () { g; }
g () { a=bar; }
a=foo
f
echo $a      ## echoes "bar"
```

- Example 2-in-1 (expansion and dynamic scoping):

```
f () { echo $1 $a; }
a=foo
a=bar f $a      ## echoes "foo bar"
echo $a         ## echoes "bar"
```

# Examples – Let's have fun with behaviours

- Nice falses and the violent one:

```
false && true
! true
false
```

# Examples – Let's have fun with behaviours

- Nice falses and the violent one:

```
false && true
! true
false
```

-

```
( exit )
( return )
exit | true
exit
```

# Basic definitions

| | | | |
|---|---|---|---|
| String variables | $x_s$ | $\in$ | $SVar$ |
| List variables | $x_l$ | $\in$ | $LVar$ |
| Procedures | $c$ | $\in$ | $\mathcal{F}$ |
| Natural numbers | $n$ | $\in$ | $\mathbb{N}$ |
| Strings | $\sigma$ | $\in$ | $String$ |

| | | | |
|---|---|---|---|
| Programs | $p$ | $::=$ | $vdecl^*$ pdecl$^*$ program t |
| Variables declarations | $vdecl$ | $::=$ | **varstring** $x_s$ $\mid$ **varlist** $x_l$ |
| Procedures declarations | $pdecl$ | $::=$ | **proc** $c$ **is** $t$ |

# Terms and expressions

| | | | |
|---|---|---|---|
| String expressions | $s$ | $::=$ | $f_{\mathsf{s}}^*$ |
| String fragments | $f_{\mathsf{s}}$ | $::=$ | $\sigma \mid x_{\mathsf{s}} \mid n \mid t$ |
| List expressions | $l$ | $::=$ | $f_{\mathsf{l}}^*$ |
| List fragments | $f_{\mathsf{l}}$ | $::=$ | $[s] \mid \textbf{split}\ s \mid x_{\mathsf{l}}$ |
| Terms | $t$ | $::=$ | $\textbf{true} \mid \textbf{false} \mid \textbf{fatal}$ |
| | | $\mid$ | $\textbf{return}\ t \mid \textbf{exit}\ t$ |
| | | $\mid$ | $x_s := s \mid x_l := l$ |
| | | $\mid$ | $t\ ;\ t \mid \textbf{if}\ t\ \textbf{then}\ t\ \textbf{else}\ t$ |
| | | $\mid$ | $\textbf{for}\ x_s\ \textbf{in}\ l\ \textbf{do}\ t \mid \textbf{do}\ t\ \textbf{while}\ t$ |
| | | $\mid$ | $\textbf{process}\ t \mid \textbf{pipe}\ t\ \textbf{into}\ t$ |
| | | $\mid$ | $\textbf{call}\ l \mid \textbf{shift}$ |

# Basic definitions

| | | | |
|---|---|---|---|
| Strings | $\sigma$ | $\in$ | $String$ |
| Lists | $\lambda$ | $\in$ | $StringList \triangleq \{ \sigma^* \mid \sigma \in \mathsf{String} \}$ |
| Term behaviours | $b$ | $\in$ | $\{\mathsf{True}, \mathsf{False}, \mathsf{Fatal}, \mathsf{Return\ True}$ $\mathsf{Return\ False}, \mathsf{Exit\ True}, \mathsf{Exit\ False}\}$ |
| Expression behaviours | $\beta$ | $\in$ | $\{\mathsf{True}, \mathsf{Fatal}, \mathsf{None}\}$ |
| File systems | | | $\mathcal{FS}$ |
| String environments | | | $SEnv \triangleq [\mathsf{SVar} \rightarrow \mathsf{String}]$ |
| List environments | | | $LEnv \triangleq [\mathsf{LVar} \rightarrow \mathsf{StringList}]$ |
| Contexts | $\Gamma$ | $\in$ | $\mathcal{FS} \times \mathsf{String} \times \mathsf{StringList}$ $\times\ \mathsf{SEnv} \times \mathsf{LEnv}$ |

# Judgements

| | | | |
|---|---|---|---|
| Terms | $t_{/\Gamma}$ | $\Rightarrow$ | $\sigma \star b_{/\Gamma'}$ |
| String fragments | $f_{s\,/\Gamma}$ | $\rightsquigarrow_{sf}$ | $\sigma \star \beta_{/\Gamma'}$ |
| String expressions | $s_{/\Gamma}$ | $\rightsquigarrow_{s}$ | $\sigma \star \beta_{/\Gamma'}$ |
| List fragments | $f_{l\,/\Gamma}$ | $\rightsquigarrow_{lf}$ | $\lambda \star \beta_{/\Gamma'}$ |
| List expressions | $l_{/\Gamma}$ | $\rightsquigarrow_{l}$ | $\lambda \star \beta_{/\Gamma'}$ |

# if

$$\frac{t_{1/\Gamma} \Rightarrow \sigma_1 \star \mathsf{True}_{/\Gamma_1} \qquad t_{2/\Gamma_1} \Rightarrow \sigma_2 \star b_{2/\Gamma_2}}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_{3/\Gamma} \Rightarrow \sigma_1\sigma_2 \star b_{2/\Gamma_2}} \text{ If-True}$$

$$\frac{\begin{array}{c} t_{1/\Gamma} \Rightarrow \sigma_1 \star b_{1/\Gamma_1} \qquad b_1 \in \{\mathsf{False}, \mathsf{Fatal}\} \\ t_{3/\Gamma_1} \Rightarrow \sigma_3 \star b_{3/\Gamma_3} \end{array}}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_{3/\Gamma} \Rightarrow \sigma_1\sigma_2 \star b_{2/\Gamma_2}} \text{ If-False}$$

$$\frac{t_{1/\Gamma} \Rightarrow \sigma_1 \star b_{1/\Gamma_1} \qquad b_1 \in \{\mathsf{Return}\ \_, \mathsf{Exit}\ \_\}}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_{3/\Gamma} \Rightarrow \sigma_1 \star b_{1/\Gamma_1}} \text{ If-Transmit}$$

# String fragments

$$\frac{}{\sigma_{/\Gamma} \rightsquigarrow_{sf} \sigma \star \mathrm{None}_{/\Gamma}}\ \text{StrFrag-Litteral}$$

$$\frac{}{x_{s\,/\Gamma} \rightsquigarrow_{sf} \Gamma.senv[x_s] \star \mathrm{None}_{/\Gamma}}\ \text{StrFrag-Variable}$$

$$\frac{}{n_{/\Gamma} \rightsquigarrow_{sf} \Gamma.args[n] \star \mathrm{None}_{/\Gamma}}\ \text{StrFrag-Argument}$$

$$\frac{t_{/\Gamma} \Rightarrow \sigma \star b_{/\Gamma'}}{t_{/\Gamma} \rightsquigarrow_{sf} \sigma \star \overline{b}_{/\Gamma'}}\ \text{StrFrag-Term}$$

# String expressions

$$\frac{}{\varepsilon_{s\,/\Gamma} \rightsquigarrow_s \text{“”} \star \text{None}_{/\Gamma}}\ \text{Str-Empty}$$

$$\frac{f_{s\,/\Gamma} \rightsquigarrow_{sf} \sigma \star \beta_{\,/\Gamma'} \qquad s_{/\Gamma'} \rightsquigarrow_s \sigma' \star \beta'_{\,/\Gamma''}}{f_s s_{/\Gamma} \rightsquigarrow_s \sigma \cdot \sigma' \star \beta\beta'_{\,/\Gamma''}}\ \text{Str-NonEmpty}$$

# List fragments

$$\frac{s_{/\Gamma} \rightsquigarrow_s \sigma \star \beta_{/\Gamma'}}{[s]_{/\Gamma} \rightsquigarrow_{lf} [\sigma] \star \beta_{/\Gamma'}} \text{ LstFrag-Singleton}$$

$$\frac{s_{/\Gamma} \Rightarrow \sigma \star \beta_{/\Gamma'}}{\textbf{split}\ s_{/\Gamma} \rightsquigarrow_{lf} \textbf{split}\ \sigma \star \beta_{/\Gamma'}} \text{ LstFrag-Split}$$

$$\frac{}{x_{l\,/\Gamma} \rightsquigarrow_{lf} \Gamma.lenv[x_l] \star \text{None}_{/\Gamma}} \text{ LstFrag-Variable}$$

# List expressions

$$\frac{}{\varepsilon_{I/\Gamma} \rightsquigarrow_I [] \star \mathsf{None}_{/\Gamma}}\ \text{Lst-Empty}$$

$$\frac{f_{I/\Gamma} \rightsquigarrow_{If} \lambda \star \beta_{/\Gamma'} \qquad I_{/\Gamma'} \rightsquigarrow_I \lambda' \star \beta'_{/\Gamma''}}{f_I I_{/\Gamma} \rightsquigarrow_I \lambda +\!\!+ \lambda' \star \beta\beta'_{/\Gamma''}}\ \text{Lst-NonEmpty}$$