Command injection attacks, continuations, and the Lambek calculus

Hayo Thielecke
University of Birmingham

Overview

Aim: to connect

1. Command injection attacks;
2. continuations and control effects;
3. the Lambek calculus, a presentation of syntax as a logic or type system

- No theorems
- Some intuitions (I hope)

Continuations

Consider an expression language with a control operator:

$$\begin{array}{rcl}
[\![E_1 + E_2]\!] & = & \lambda k.[\![E_1]\!](\lambda x_1.[\![E_2]\!](\lambda x_2.k(x_1 + x_2)) \\
[\![n]\!] & = & \lambda k.k\,n \\
[\![\mathtt{return}\ n]\!] & = & \lambda k.n
\end{array}$$

For example, the expression

$$(\mathtt{return}\ 42) + 666$$

evaluates to 42. The continuation $(\bigcirc + 666)$ has been discarded.
The typing of the continuation semantics is a double negation:

$$[\![E]\!] : (\mathtt{int} \to \mathtt{int}) \to \mathtt{int}$$

Continuations and generalized double negation

Intuitionistic negation:

$$\neg\tau = \tau \to \bot$$

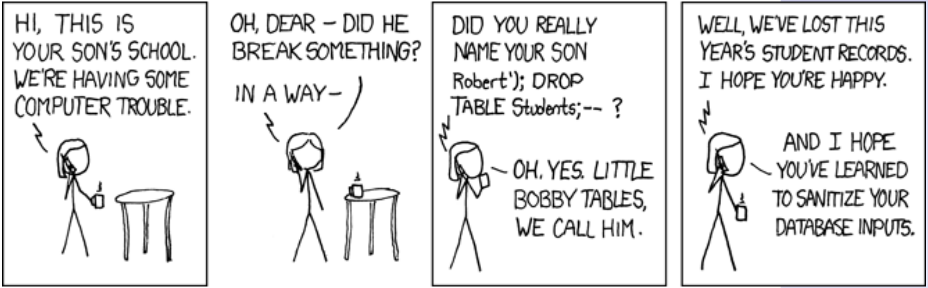
$$[\![E]\!] : (\mathtt{int} \to \mathtt{int}) \to \mathtt{int}$$

is a generalized double negation using

$$\neg\tau = \tau \to \mathtt{int}$$

Other generalizations are possible:
$\multimap$ instead of one of both $\to$: linear continuation passing
$\neg\tau = \tau \to \alpha$: answer type polymorphism
In this talk: two different arrows, $\searrow$ and $\swarrow$ instead of $\to$

SQL injection attack



Source: `http://xkcd.com/327/`

Malicious input:

`Robert '); DROP TABLE Students; --`

Side effects vs syntactic effects

- `DROP Table` needs side-effects
- But: `OR 1 = 1` does not; purely functional language
- String with a hole
- compare: expression with a hole for control operators
- not a side effect in the sense of state
- but still an effect: a control effect, in syntax
- make this precise: as a double negation type

Tautology attack

Purely functional language of Boolean expressions

String with a hole: `password = ◯`

Legitimate input: `foo`

Combined string: `password = foo`

Malicious input: `foo OR 1 = 1`

Combined string: `password = foo OR 1 = 1`

Parsed like: `(password = foo) OR (1 = 1)`

Evaluates to: `true`

This is not just plugging the ◯; something has happened: an effect.

What next: we need a calculus for syntax

In the control operator example, there was a term with a hole

$$(\bigcirc + 666)$$

In the tautology injection example, there was a string with a hole

$$\texttt{password = } \bigcirc$$

For control operators, we have generalized double negation

$$((-) \to \texttt{int}) \to \texttt{int}$$

For strings, we need to generalize the double negation even more:
Lambek calculus with two arrows $\searrow$ and $\swarrow$.

Lambek calculus

- Lambek's syntactic calculus
- "The mathematics of sentence structure" (1958)
- A type system or logic for syntax
- Builds on older ideas from mathematical logic and linguistics
- Mainly used in linguistics
- Substructural calculus
- Very simple model in terms of strings
- Forerunner of:
  - Linear logic
  - Separation logic

Hayo Thielecke
University of Birmingham

Left and right arrows example: infix operator OR

Using a grammar:

$$T ::= T \; \texttt{OR} \; T$$

Using arrows:

$$\texttt{OR} \lhd (T \searrow T) \swarrow T$$

Partially applied, still expecting something on the left:

$$\texttt{OR 1 = 1} \lhd T \searrow T$$

Fully applied:

$$\texttt{1 = 0 OR 1 = 1} \lhd T$$

Substructural logics

Typical rule in logic and type theory:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \ (\to \mathrm{I})$$

In logic, there are substructural rules:

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A} \ (\mathrm{Weakening})$$

$$\frac{\Gamma, B, B \vdash A}{\Gamma, B \vdash A} \ (\mathrm{Contraction})$$

$$\frac{B, C \vdash A}{C, B \vdash A} \ (\mathrm{Exchange})$$

In a substructural logics, some or all of there rules are absent.

Two implications instead of one

Lambda calculus:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x . M : A \to B} \, (\to \mathrm{I})$$

Lambek calculus:

$$\frac{\varphi \; \Phi \lhd \psi}{\Phi \lhd \varphi \searrow \psi} \, (\searrow \mathrm{R}) \qquad \frac{\Phi \; \varphi \lhd \psi}{\Phi \lhd \psi \swarrow \varphi} \, (\swarrow \mathrm{R})$$

Semantics as sets of strings

$$
\begin{array}{rcl}
[\![X]\!] & = & \{w \in \mathbf{T}^* \mid X \stackrel{*}{\Rightarrow} w\} \\
[\![\varphi \searrow \psi]\!] & = & \{w \in \mathbf{T}^* \mid \forall v \in \mathbf{T}^*. v \in [\![\varphi]\!] \text{ implies } v\, w \in [\![\psi]\!]\} \\
[\![\psi \swarrow \varphi]\!] & = & \{w \in \mathbf{T}^* \mid \forall v \in \mathbf{T}^*. v \in [\![\varphi]\!] \text{ implies } w\, v \in [\![\psi]\!]\} \\
[\![\varphi_1 \circ \varphi_2]\!] & = & \{w_1\, w_2 \in \mathbf{T}^* \mid w_1 \in [\![\varphi_1]\!] \text{ and } w_2 \in [\![\varphi_2]\!]\} \\
[\![\varepsilon]\!] & = & \{\varepsilon\}
\end{array}
$$

Sequent version of the Lambek calculus

$$\frac{\Phi \lhd \varphi \qquad \Psi\ \psi\ \Pi \lhd \pi}{\Psi\ (\psi \swarrow \varphi)\ \Phi\ \Pi \lhd \pi}\ (\swarrow\mathrm{L}) \qquad \frac{\Phi\ \varphi \lhd \psi}{\Phi \lhd \psi \swarrow \varphi}\ (\swarrow\mathrm{R})$$

$$\frac{\Phi \lhd \varphi \qquad \Psi\ \psi\ \Pi \lhd \pi}{\Psi\ \Phi\ (\varphi \searrow \psi)\ \Pi \lhd \pi}\ (\searrow\mathrm{L}) \qquad \frac{\varphi\ \Phi \lhd \psi}{\Phi \lhd \varphi \searrow \psi}\ (\searrow\mathrm{R})$$

$$\frac{\Phi\ \varphi\ \psi\ \Psi \lhd \pi}{\Phi\ (\varphi \circ \psi)\ \Psi \lhd \pi}\ (\circ\mathrm{L}) \qquad \frac{\Phi \lhd \varphi \qquad \Psi \lhd \psi}{\Phi\ \Psi \lhd \varphi \circ \psi}\ (\circ\mathrm{R})$$

$$\frac{\Phi \lhd \varphi \qquad \Psi\ \varphi\ \Pi \lhd \psi}{\Psi\ \Phi\ \Pi \lhd \psi}\ (\mathrm{CUT}) \qquad \frac{}{\varphi \lhd \varphi}\ (\mathrm{AX})$$

Double negations

$$\frac{\Phi \lhd \varphi}{\Phi \lhd (\psi \swarrow \varphi) \searrow \psi}\ (\text{DNIL}) \qquad \frac{\Phi \lhd \varphi}{\Phi \lhd \psi \swarrow (\varphi \searrow \psi)}\ (\text{DNIR})$$

Intuitively:
Suppose we have a φ.
Then if there is a

$$(\psi \swarrow \varphi)$$

to the left of the φ, we can get a ψ.
So we have a

$$(\psi \swarrow \varphi) \searrow \psi$$

Contrast the arrows with:

$$\mathtt{OR} \lhd (T \searrow T) \swarrow T$$

Tautology injection in the Lambek calculus

The malicious inputs have a double negation type:

String	has type	fitting into context
`b OR 1 = 1`	$(T \swarrow V) \searrow E$	`a =` ◯
`1 = 1 OR b`	$E \swarrow (V \searrow T)$	◯ `= a`

Lambek's examples from linguistics

The pronoun "he" must be to the left of the verb; "him" must be to the right.

String	has type	fitting into context
`he`	$\mathbf{Sen} \swarrow (\mathbf{Noun} \searrow \mathbf{Sen})$	$\bigcirc$ `knows Alice`
`him`	$(\mathbf{Sen} \swarrow \mathbf{Noun}) \searrow \mathbf{Sen}$	`Alice knows` $\bigcirc$

Double negation in control and command injection

Control operators:

```
return 42
```

in direct style, typed as `int`.
But semantically, a double negation of an `int`

$$(\texttt{int} \to \texttt{int}) \to \texttt{int}$$

Command injection:

```
1 = 1 OR b
```

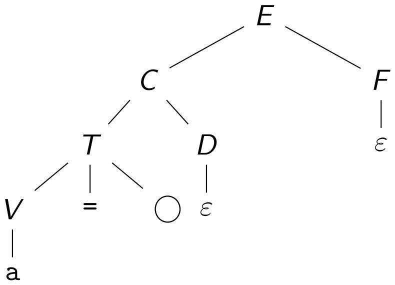
plugged into a context expecting a string.
But actually, a double negation of a string,

$$E \swarrow (V \searrow T)$$

Parse tree surgery by syntactic effects

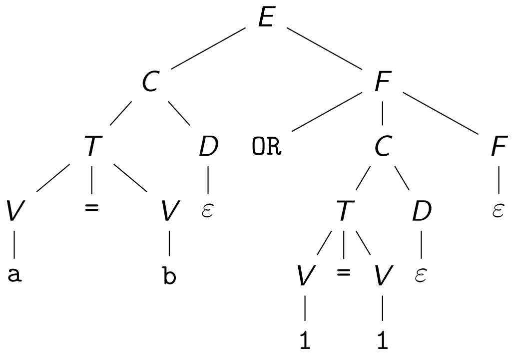
- Command injection attacks can also be understood in terms of parse trees.
- Connection to Lambek calculus: double negated types fit into a tree with a hole
- BUT: they do not stay inside the hole
- Instead: tree surgery
- Compare: expression tree manipulation by control operators

Parse tree with a hole



Hayo Thielecke
University of Birmingham

Parse tree after tautology injection



Hayo Thielecke
University of Birmingham

Security as malicious versions of computer science

- Computer security
  $=$ Satan's computer
- Command injection attacks
  $=$ Satan's interpreter
- Command injection without side effects
  $=$ Satan's parser
- Classic buffer overflow overwriting return address
  $=$ Satan's continuation passing
- Advanced buffer overflow with return-oriented programming
  $=$ Satan's combinatory logic $+$ continuation passing

Related work

- Command injection attack defences (Su and Wasserman)
- Grammar-based program analysis (Thiemann)
- Continuations in linguistics (Barker, Shan)
- Semantics of parsing actions (HT)

Directions for future work:

- Are syntactic command injections always characterised by double negations?
- Formally connect control operators and syntax?
- Correctness of program analysis via Lambek calculus?
- The Lambek calculus is cool.